VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

RETROSPEKTIVNÍ HRA PRO DVA HRÁČE S VESMÍRNOU TÉMATIKOU

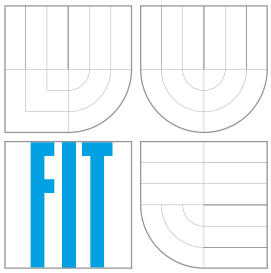
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ KIMER

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

RETROSPEKTIVNÍ HRA PRO DVA HRÁČE S VESMÍRNOU TÉMATIKOU

RETROSPECTIVE SPACE GAME FOR TWO PLAYERS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ KIMER

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR POSPÍCHAL

BRNO 2011

Abstrakt

Tato bakalářská práce se zabývá vývojem počítačové hry inspirované jednou z prvních počítačových her vůbec – Spacewar!. První část popisuje počítačové hry a s nimi spojenou historii, současný stav, jejich žánry a hry retrospektivní. Druhá část se věnuje hardwarově akcelerované počítačové grafice, zaměřuje se na její historii a rozebírá grafické knihovny, především knihovnu OpenGL. Další část práce se věnuje návrhu vyvíjené hry, řeší pohyb objektů, kolize mezi nimi, efekty pomocí částicových systémů, ovládání pro dva hráče na jednom počítači a základní grafické uživatelské rozhraní. Následuje popis implementace v jazyce C++ s využitím knihovny OpenGL a rozšiřující knihovny SDL za použití objektového návrhu. Závěrečná část je věnována testování výsledné aplikace včetně analýzy hodnocení uživatelů.

Abstract

This bachelor's thesis describes the development of a computer game inspired by one of the first computer games ever – Spacewar!. The first part describes computer games and associated history, current status, their genres and retrospective games. The second part deals with hardware-accelerated computer graphics, focused on its history and discusses the graphics libraries, especially OpenGL. Next part is devoted to design of the developed game, addressing the movement of objects, collisions between them, effects using particle systems, controls for two players on one computer and a basic graphics user interface. Next chapter covers implementation in C++ language using OpenGL and SDL libraries and object oriented design. The final section is devoted to testing the resulting application, including analysis of user ratings.

Klíčová slova

Počítačová hra, Spacewar!, Hardwarově akcelerovaná počítačová grafika, OpenGL, Kolize, Částicový systém.

Keywords

Computer game, Spacewar!, Hardware-accelerated computer graphics, OpenGL, Collision, Particle system.

Citace

Tomáš Kimer: Retrospektivní hra pro dva hráče s vesmírnou tématikou, bakalářská práce, Brno, FIT VUT v Brně, 2011

Retrospektivní hra pro dva hráče s vesmírnou tématikou

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Petra Pospíchala

.....
Tomáš Kimer
18. května 2011

Poděkování

Tímto bych rád poděkoval Ing. Petru Pospíchalovi za cenné rady, ochotu a dobrou motivaci. Díky patří i rodičům za jejich podporu při studiu.

Obsah

Kapitola 1

Úvod

V dnešní době jsou počítačové hry velmi rozsáhlým a rychle se vyvíjejícím odvětvím počítačového průmyslu. O hry je velký zájem a mnoho lidí si bez nich život už nedokáže představit. Proto se vyvíjejí stále nové a nové tituly, které jsou čím dál propracovanější – ať už ze strany vzhledu, anebo ze strany obsahu.

Lidé, kteří se baví počítačovými hrami již od dob jejich rozvoje, si v dnešní době plné propracovaných titulů rádi zavzpomínají na jednoduché hry svého mládí, u kterých strávili mnoho hodin dobré zábavy. Velký počet těchto her ale již není na dnešní počítače dostupných. Proto se objevují různé konverze nebo nové implementace, které hru můžou navíc vzhledově či hratelnostně vylepšit. Zábavný koncept je pak zachován, ale hratelnost je zpestřena díky většímu výkonu dnešních počítačů.

Tato práce se zabývá tvorbou retrospektivní počítačové hry inspirované hrou Spacewar!, jejíž základní koncept je znám prakticky od počátku herní historie – vzájemný souboj dvou hráčů na jednom počitači, kteří ovládají každý svou vesmírnou loď a na ploše jedné obrazovky se snaží pomocí různých zbraní zničit loď hráče druhého [19]. Hře bude zachován retrospektivní nádech a její základní koncept bude obohacen o různé další vizuální a hratelnostní prvky.

1.1 Členění práce

Práce je členěna do několika kapitol.

V úvodní kapitole je rozebrán pojem počítačových her – od jejich žánrů, přes historii, až po současnost. Zároveň jsou diskutovány retrospektivní hry, zejména vesmírného žánru.

Následující, 3. kapitola probírá hardwarově akcelerovanou počítačovou grafiku – její rozvoj a grafické knihovny. Jsou zde popsány knihovny OpenGL a DirectX.

Kapitola 4 rozebírá konkrétní návrh řešení aplikace – základní koncept, volbu platformy, návrh herních prvků, jejich pohyb a interakce, efekty s použitím částicových systémů, ovládání pro dva hráče a grafické uživatelské rozhraní. Pozornost je také věnována detekci kolizí.

V další části (kapitola 5) je popis samotné implementace – architektura aplikace, její objektový návrh, popis tříd a informace o překladu.

Předposlední, 6. kapitola obsahuje testování implementovaného systému včetně analýzy hodnocení uživatelů.

Závěrečná část zhodnocuje dosažené výsledky a diskutuje možnosti dalšího rozšíření.

Kapitola 2

Počítačové hry

Touha hrát je stará jako lidstvo samo. S rozvojem počítačových her dostalo hraní nový rozměr – člověk se za obrazovkou monitoru může stát prakticky kýmkoliv a vyzkoušet si prakticky cokoliv. Díky tomu se staly počítačové hry velice oblíbenými.

Počítačová hra je softwarový systém, který je provozován na počítači, nejčastěji osobním. Hry je možno hrát i na tzv. *konzolích*, což jsou zařízení určená pouze pro hraní her, a to především s využitím televizni obrazovky (nazýváme pak termínem *videohra*).

Hra se obvykle skládá z virtuálního prostředí a hráč s ní interakuje (ovlivňuje dění) pomocí klávesnice, myši nebo jiného ovladače, a plní různé cíle. Hra reflektuje hráčovy akce změnou grafiky na obrazovce, přehráváním zvuků, hudby, či dokonce vibracemi ovládacích prvků. Existuje velké množství rozličných herních žánrů. Hry můžou vyprávět i příběh, kterým hráče vtáhnou do děje.

2.1 Žánry počítačových her

Počítačové hry se už od začátku historie dělí na různé žánry, díky kterým lze jednoduše identifikovat, co může hráč od hry očekávat. Existuje i množství her, které kombinují více žánrů. Jednotlivé žánry jsou následující:

Adventura[1] je žánr vyprávějící příběh, ve kterém hráč ovládá postavu řešící různé logické úkoly a hádanky, a to často pomocí vhodné kombinace předmětů, které nachází po herním světě. Souboje jsou v těchto hrách jen vyjímečně.

Akční hry spočívají v soubojích s nepřáteli, obsahují násilné scény a množství zbraňového arzenálu. Častým zástupcem jsou trojrozměrné střílecí hry viděné z vlastního (prvního) pohledu (*First-Person Shooter*, zkratka FPS). Úkolem je většinou prostřílet se velkým počtem nepřátel a dostat se do další úrovně. Pokud je hra pouze ve dvou rozměrech, setkáme se s označením *plošinová* hra. Speciálním případem jsou bojové hry, ve kterých jsou do arény umístěni dva protivníci a jediným úkolem je přemoci toho druhého [16].

Arkáda je žánr založený na jednoduchém a nápaditém konceptu – dříve velmi rozšířený na herních automatech.

Hra na hrdiny (*Role-Playing Game*, zkratka RPG) je žánr, ve kterém hráč ovládá svého hrdinu, se kterým chodí po herním světě, plní různé úkoly a postupně získává nové

[1] Podle anglického slova *adventure* – dobrodružství.

schopnosti a vlastnosti. Oproti adventurám dochází k mnoha soubojům. Speciálním případem je hra pro více hráčů (*Massively Multiplayer Online Role-Playing Game*, zkratka MMORPG), ve které spolu hráči hrají ve velkém, virtuálním, perzistentním světě skrze internet.

V **logické** hře je hráčovým úkolem pouze řešit více či méně náročné logické problémy, například hlavolamy apod.

Online hra je žánr cílený pouze na hraní po internetu s ostatními hráči. Tento žánr existuje ve spojitosti s dalšími žánry, například hry na hrdiny.

Simulátor je typ hry, která reálně simuluje nejčastěji nějaký dopravní prostředek, například letadlo, loď či závodní auto. Věrnost simulace se u různých her liší, ale může být i velmi přesná.

Sportovní hry nechávají hrače vžít se do sportovního zážitku, například ovládání hráčů při fotbalovém, hokejovém či tenisovém zápasu, lovení zvěře v přírodě apod. Lze je označit jako speciální případ žánru simulace.

Strategie je žánr, kde hráč ovládá větší množství objektů (jednotek) viděných z nadhledu a snaží se je správnými příkazy ovládat tak, aby splnil nějaký, předem daný cíl. Existuje více specializací, například tahové, kde v jednu dobu hraje současně jen jeden hráč (střídají se na tahy), strategie v reálném čase (*real-time*), kdy hrají oba hráči současně, anebo manažerské strategie, ve kterých je hráčovým úkolem vybudovat co nejlepší město, nemocnici, či zábavní park [16].

V **závodních** hrách hráč nejčastěji ovládá nějaký dopravní prostředek, s kterým se účastní závodů a snaží se dojet do cíle jako první. Můžeme jej také označit jako speciální případ žánru simulací, vetšinou s méně reálnou věrností.

2.2 Minulost

Tato podkapitola zabývající se minulostí počítačových her čerpá především z článku [19] a knihy [28].

2.2.1 50. a 60. léta – počátky

První jednoduché, leč zábavné programy připomínající dnešní hry se začaly objevovat již na počátku 50. let 20. století. Jedna z prvních předchůdců počítačových her byla hříčka *Tennis for Two*, která byla vytvořena na analogovém počítači s použitím osciloskopu jako zobrazovacího zařízení. V roce 1958 ji v laboratoři vytvořil William Higinbotham jako experiment demonstrující interaktivní ovládání. Hra z profilu zobrazovala tenisový kurt, obsahovala síť i míček a byla ovládána dvěma krabičkami s tlačítkem pro odpal a kolečkem pro nastavení trajektorie míčku.

První opravdová počítačová hra vznikla o pár let později, začátkem 60. let. V roce 1962 studenti Massachusettského technologického institutu (MIT) vytvořili na stroji *DEC PDP-1* hru *Spacewar!* (obrázek 2.1, převzat z [25]). Hra byla určena pro dva hráče, z nichž každý ovládal svou vesmírnou loď a snažil se pomocí zbraně zničit toho druhého. Uprostřed mapy byla hvězda, která na lodě působila gravitací. Tento koncept hry se dočkal mnoha následovníků, kteří jej různě vylepšovali a je znám prakticky dodnes. Je také inspirací této práce.

Obrázek 2.1: Hra Spacewar! na stroji DEC PDP-1 (1962).

2.2.2 70. léta – herní automaty

Opravdový začátek pro videohry znamenala až 70. léta. Nolan Bushnell se inspiroval hrou Spacewar! a vytvořil první hru na herním automatu – *Computer Space* (1971). O rok později založil společně s Tedem Dabneyem společnost Atari a ve stejný rok vydali první masově rozšířenou hru, *Pong*. Tato arkádová hra připomínající stolní tenis je pro počátek videoher tak typická, že je mnohými považována za úplně tu první. Jako další z arkádových her vyšla roku 1978 hra *Space Invaders* od japonské společnosti Taito. Podle Guinessovy knihy rekordů se stala nejúspěšnější arkádovou hrou – v Japonsku po jejím uvedení dokonce způsobila dočasný nedostatek mincí. Roku 1980 vyšla další z legendárních her – *Pac-Man* společnosti Namco. Hra se stala symbolem nového žánru po celém světě.

Vzhled všech těchto doposud uvedených her nebyl nijak převratný – vše se skládalo pouze ze základních geometrických primitiv, jako je úsečka, polygon či elipsa. Počet barev byl také omezen – v úplných začátcích pouze dvě, nejčastěji bílá a černá (pozadí). Grafika se ale začala postupně zlepšovat.

2.2.3 80. léta – domácí herní systémy

Osmdesátá léta se nesla ve znamení řady úspěšných domácích herních systémů – *Apple II*, *Atari*, *IBM PC*, *Sinclair ZX Spectrum*, *Commodore 64* či *Amiga*. Zařízením postupně rostl výkon a současně se začlo rozvíjet mnoho nových herních žánrů, což vedlo k rostoucí popularitě her. *Defender* (1980) přišel s žánrem střílečka s horizontálně posouvaným obrazem, *Battlezone* (1980) využívala vektorové grafiky k prvnímu skutečnému trojrozměrnému zobrazení herního světa, *3D Monster Maze* (1982) byl první trojrozměrnou hrou pro do-

mácí počítače, závodní *Pole Position* (1982) disponovalo pseudo-3D[2] grafikou založenou na spritech[3]) a *Dragon's Lair* (1983) byla první hra na optickém disku a využívající filmové animace – podobně jako *Phantasmagoria* na PC. Velký význam měl vzestup žánru adventur. *King's Quest: Quest for the Crown (1984)* od společnosti Sierra byl v adventurách revolucí, jelikož nabízel pseudo-3D prostředí s plně animovaným pohybem postaviček, stejně jako dalších objektů, ovládanými kurzorovými šipkami.

V osmdesátých letech přišel také rozvoj *handheldů* – přenosných herních krabiček. Jejich velké rozšíření bylo zapříčiněno jedinou hrou – *Tetris* (1985). Roku 1989 vyšel na dlouhou dobu nejtypičtější zástupce handheldů – *Nintendo Game Boy*.

2.2.4 90. léta – postupné rozšiřování a 3D akcelerace

V 90. letech se hry postupně začaly stávat mainstreamovou zábavou a zvýšily se rozpočty herního trhu. To bylo způsobeno technologickým rozvojem, díky kterému hry mohly vypadat lépe a věrohodněji. Roku 1992 pokládá hra *Dune II* základ žánru strategických her v reálném čase – není zdaleka první, ale všechny další hry tohoto žánru na ni staví. Vycházejí hry jako *Warcraft: Orcs & Humans* či *Command & Conquer*. Vychází také hospodářské strategie, jejichž základ položila hra *SimCity* (1989).

Roku 1996 se objevil první dostupný 3D akcelerátor pro domácí počítače – *3dfx Voodoo*. Jeho výkonu začaly především využívat střílečky z prvního pohledu (FPS). *Quake* přinesl na svou dobu revoluční 3D engine[4] – umožňoval tvorbu skutečných trojrozměrných prostor[5] a dokázal využít akcelerace. S rozvojem výkonu ve 3D zpracování začal třetí rozměr pronikat do většiny her. Jsou vyvíjeny stále propracovanější 3D enginy, které společně s novými typy grafických akcelerátorů podporují stále nové a nové efekty. Tento trend se drží až dodnes.

2.2.5 Nedávná minulost – hry více hráčů a fyzika

S rozvojem internetu se začaly rozšiřovat hry pro více hráčů (multiplayer) – objevují se čistě multiplayerové hry, jako *Counter-Strike* (1999) či *Quake III Arena* (2003). Také roste obliba MMORPG, které hrají stovky až tisíce hráčů současně. Jednou z prvních her byla *Ultima Online* (1997). Roku 2004 vyšel *World of Warcraft*, který je v oblibě až do současnosti a nedávno (2010) překročil hranici dvanácti miliónů aktivně hrajících hráčů [15]. Zvláštní kapitolou jsou potom hry na sociálních sítích, jejichž nejznámější zástupce *FarmVille* dosáhl v září roku 2010 62 miliónů aktivních uživatelů[6].

Vývojáři se, kromě grafické stránky, začínají věnovat také podpoře fyziky – vznikají první fyzikální enginy, jako je například *Havok*, který jako jedna z prvních her využila hra *Max Payne 2* (2003). V současnosti se výpočty fyziky začínají akcelerovat na výkonných grafických kartách. Hry se stávají stále propracovanějšími.

[2] Pseudo-3D (někdy označované jako 2.5D) označuje dvourozměrnou grafiku, která se pomocí různých technik snaží dosáhnout trojrozměrného dojmu.

[3] Sprite je v počítačové grafice malý dvourozměrný obrázek, který je integrován do větší scény.

[4] 3D engine je softwarový systém pro vykreslování trojrozměrné počítačové grafiky.

[5] Skutečných proto, protože do této doby hry neuměly pracovat například s dvěmi chodbami umístěnými nad sebou.

[6] Viz http://www.facebook.com/FarmVille.

2.3 Současný stav

V současné době jsou počítačové hry už neodmyslitelnou součástí našeho života. Bez osobního počítače se v dnešní době člověk už skoro neobejde a nějakou hru si na něm zahrál skoro každý. Díky dostupnosti už hry nejsou výsadou jen pár skalních příznivců. O hry je velký zájem a z vývoje her se stal dobrý byznys. Stále rostou požadavky na vzhledově kvalitnější, propracovanější a realističtější hry. Díky tomu jsou hlavně hry hnacím motorem technologického pokroku hardwaru.

Za téměř 50 let vývoje hry prošly obrovským pokrokem – od jednoduché čárové grafiky, přes první 3D, až po fotorealisticky vypadající hry s reálnou fyzikou, poutavým příběhem, kvalitní hudbou a velkou rozsáhlostí, které si nezadají ani se zážitkem z celovečerního filmu. Není divu, jelikož dnešní hry jsou dokonce tak složité, že jejich vývoj je často náročnější než produkce takového filmu [17].

Objevují se také nové trendy v ovládání her, místo tradičních ovladačů například pomocí pohybu těla (například Microsoft *Kinect*[7]) či hlasu. Dalo by se říct, že pokud pokrok půjde stejným tempem, můžeme se v budoucnu těšit na velmi zajímavé věci.

2.4 Retrospektivní hry

Starší hráči, kteří vyrůstali s rozvojem her, se rádi vrací ke hrám svého mládí, ať už z nostalgie, anebo proto, že je prostě dobře bavily. V dnešní době už je ale toto problém, jelikož počítače pokročily, a mnoho her už na nich nejde zpětně spustit, případně je dohledat. Původní autoři už nemají zájem své staré hry předělávat. Proto vznikají různé předělávky těchto her (většinou vyvinuty zdarma nadšenci), které navíc hru při zachování původního konceptu ještě vylepší, čímž se stane lépe hratelnou a atraktivnější i pro mladší hráče.

2.4.1 Hry typu Spacewar!

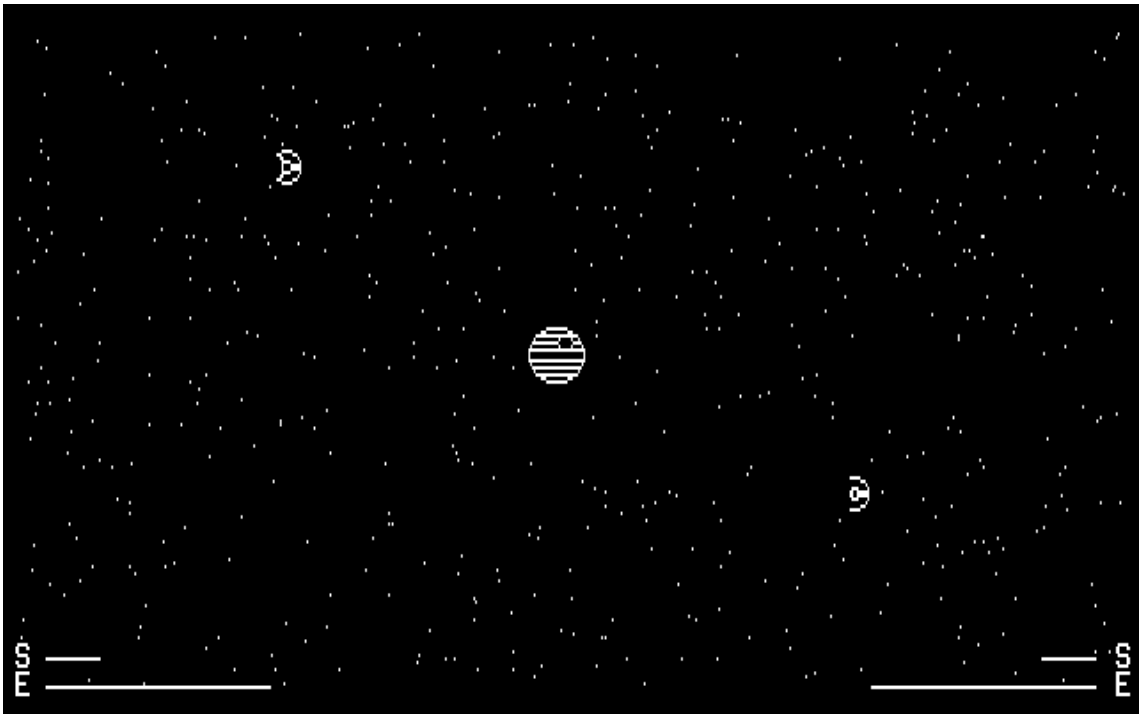
Jak už bylo zmíněno, první opravdová počítačová hra Spacewar! byla inspirací mnoha dalším, které hru více či méně vylepšovaly. Základ měly ale všechny stejný – grafika složená jen ze základních geometrických primitiv, jako je úsečka, polygon či elipsa a hra odehrávající se na ploše jedné obrazovky viděné shora. Pohyb lodí ovládaných hráči byl založen na newtonovské fyzice se setrvačností určenou směrem zážehu tryskového motoru. Tím byla stížena manipulace s loděmi, což zvyšovalo zábavnost. Při dosažení kraje obrazovky se loď ocitla na kraji opačném.

Jako první se o objevila již v části 2.2.2 zmíněná hra Computer Space, která se hrála na herním automatu a existovala hned ve dvou verzích. V jedné verzi byla jen pro jednoho hráče, který měl za úkol zničit pomocí raket dvojici nepřátelských lodí a zároveň se vyhýbat jejich střelám. Pro zničení lodě stačil jen jeden zásah. Hra měla omezení na 90 sekund a pokud hráč po uplynutí tohoto času měl větší počet bodů než počítač, hra se opakovala. V druhé verzi byla hra pro dva hráče a neobsahovala žádné, počítačem řízené, protivníky. Cílem bylo vzájemným soubojem v čase 90 sekund nasbírat co nejvíce bodů [22].

Na herních automatech či domácích herních systémech se objevilo ještě hodně podobných her (například *Space Wars* od Cinematronics roku 1977), ale první předělávkou na platformu PC (operační systém MS-DOS) se dá považovat až hra *SpaceWar* (1985, obrázek 2.2). Hru vytvořil Bill Seiler a přinášela další vylepšení. Hra byla pro dva hráče s možností zapnout jednomu hráči umělou inteligenci a hrát tak i sám proti počítači. Loď každého

[7] Viz http://www.kinect.cz/.

hráče byla chráněna štíty s pomalou regenerací a jako výzbroj sloužil laser s omezenou energií a rakety. Hráči měli možnost přesouvat energii do štítů a naopak. Planeta uprostřed hrací plochy se dala buď celá vypnout, anebo jen její gravitace. Oproti původní hře gravitace působila i na střely (u původní hry na to už nestačil výkon [19]). Hráč měl také možnost skočit do „hyperprostoru“, kdy zmizel z obrazovky a po chvíli se objevil na jiném, náhodném místě [21].



Obrázek 2.2: Hra SpaceWar (1985).

Roku 1997 vyšla hra *Space Wars* od The Plastic Factory pro systém Windows[8]. Neobsahovala ještě akcelerovanou grafiku, ale umožňovala konfiguraci většiny herních parametrů a základní efekty, například zobrazení trosek lodi při jejím zničení.

Z dalších verzí můžeme zmínit hru *KSpaceDuel* (1999), která je obsažena v mnoha Linuxových distribucích. Hra obsahovala bohatší než jen čárovou grafiku, novou zbraň – minu a možnost sbírání dvou typů bonusů – přidání energie nebo štítů[9].

Jedna z posledních verzí byla vyvinuta jako vzorový příklad v jazyce C# pro *XNA Game Studio* firmy Microsoft, které dokáže běžet jak na platformě Windows, tak i na herní konzoli *X-Box 360* téže firmy [12]. Ve hře se dá zvolit ze dvou herních režimů, *Retro* režim přesně kopíruje původní hru Spacewar!, a *Evolved* režim přidává modernější grafiku složenou z 3D texturovaných modelů, možnost nakupování nových zbraní a také volně se pohybující asteroidy po hrací ploše.

2.4.2 Další tituly

Z novějších her je v dnešní době například hodně vyvíjena konverze (port) hry *Duke Nukem 3D* (1996), která je sice hratelnostně jen kopií původní hry (díky uvolnění zdrojových

[8] Lze stáhnout z domovské stránky projektu http://www.digisys.net/users/cogs/spacewar.htm.
[9] Podrobnější informace o hře lze nalézt na http://docs.kde.org/stable/en/kdegames/kspaceduel/.

kódů), ale přídává navíc podporu 3D akcelerace, textur ve vysokém rozlišení, nových modelů a jako novinku také dynamického osvětlení, takže při zachování původní hratelnosti si svým vzhledem nezadá ani s aktuálními tituly [5]. Díky tomuto oživení se spousta hráčů k této hře vrací. Kdyby port nevznikl, pravděpodobně by se jich většina už ke hře nikdy nevrátila, jelikož původní verze byla pro MS-DOS a ten už v dnešní době není podporován. Lze sice využít emulátorů, jako je například *DOSBox*[10], ale pouze za cenu hraní v malém okně a s ne vždy dobrou funkčností a výkonem.

Existují také komerční hry, které hratelnostně přímo nekopírují nějaký vzor z minulosti, jen se jím okrajově inspirují za použití modernějšího, ale stále retrospektivního vzhledu. Například hra *Geometry Wars* (2003–2007), která vyšla na mnoho platforem a měla velký úspěch. Hráč v ní ovládal podobně jako u hry Spacewar! svou vesmírnou loď a jeho úkolem bylo přežit co nejdelší možnou dobu na hrací ploše, která se plnila množstvím různorodých nepřátel, které hráč ničil pomocí různých zbraní. Hra obsahovala pouze čárovou grafiku ve dvou rozměrech, ale pomocí akcelerovaných efektů dosahovala velmi dobrého a moderního dojmu, který ji s chytlavou hratelností zaručil úspěch [23].

2.5 Hra jako softwarový systém

Pokud se na moderní počítačovou hru podíváme z programátorského hlediska, zjistíme mnoho zajímavých věcí. Moderní počítačové hry s vysokými ambicemi na komerční úspěch jsou velmi robustní a složité systémy, které pokrývají mnohá odvětví informatiky a také dalších oborů. Proto jsou pro mnoho programátorů atraktivním tématem.

Když pomineme počítačovou grafiku, která je pro vývoj počítačových her esenciální, můžeme zmínit například odvětví simulací, umělé inteligence, počítačových sítí (hry více hráčů), optimalizací (např. vlastní správa paměti), databází, formálních jazyků (skriptování), různorodých algoritmů apod.

Z jiných odvětví stojí za zmínku fyzika, animace (včetně herců) a za vším stojící matematika (např. analytická geometrie – detekce kolizí apod.).

Z pohledu softwarového inženýrství je vývoj kvalitní, komerčně úspěšné počítačové hry velmi složitý a náročný proces, na kterém pracují několik let týmy i o stovkách členů, které můžou být dokonce rozděleny po více státech. Proto je velmi důležitý počáteční návrh, zvolení správných vývojových metodik apod. Jedním z problémů může být návratnost investic, jelikož hry se dělají především pro obyčejné lidi – špatné odhadnutí trhu může návratnost investice zničit.

Při vývoji hry je tedy třeba synchronizované práce mnoha různých odborníků – programátoři, designéři (scénaristé, designéři úrovní), grafici, animátoři, zvukaři a velké množství lidí v produkci (manažeři, producenti, projektanti a mnoho asistentů) [17]. Obvyklé je také zapojení externích konzultantů, kteří poskytují odborné informace k problému, který hra řeší (například letecký simulátor – odborník na letadla apod.).

V počátcích herní historie to měli herní vývojaři značně jednodušší. Hry většinou vyvíjeli jednotlivci a za pár týdnů bylo hotovo. Samozřejmě i dnes se dají takto vytvářet hry, ale spíše jen pro zábavu, bez větší šance na komerční úspěch [17]. I když při dobrém nápadu je i to možné, a to především na různých mobilních zařízeních.

[10] Viz http://www.dosbox.com/.

Kapitola 3

Hardwarově akcelerovaná počítačová grafika

Hardwarově akcelerovaná počítačová grafika je oblast počítačové grafiky, pro jejíž výpočty je využito speciálního grafického procesoru – akcelerátoru (*Graphics Processing Unit*, zkratka GPU). Tím je dosaženo značného urychlení, jelikož GPU je na grafické výpočty přímo specializován a hlavní procesor (*Central Processing Unit*, zkratka CPU) se může věnovat ostatním výpočtům.

Grafický procesor je obsažen v osobních počítačích obvykle na samostatné kartě, méně výkonné mohou být integrovány na základní desce, ale také přímo jako součást hlavního procesoru. V dnešní době se osobní počitače bez jakéhokoli akcelerátoru už prakticky nevyrábí – minimálně je vždy obsažen integrovaný.

Výhody akcelerace se projeví především v trojrozměrných scénách, pokročilých efektech či zpracování videa ve vysokém rozlišení, kde by bez akcelerace nebylo možno dosáhnout při zachování kvality obrazu přijatelné doby trvání výpočtu (která je zvlášť stěžejní u aplikací běžících v reálném čase, jako jsou například počítačové hry či přehrávání videa).

Nevýhodou může být u výkonějších akcelerátorů jejích cena (v řádech tisíců korun), popř. jejich spotřeba a zahřívání, což může být kritické zejména u přenosných počítačů.

3.1 Způsoby popisu počítačové grafiky

Je vhodné se pro začátek zmínit o dvou základních způsobech možnosti popisu počítačové grafiky a principech jejího zobrazení. Tato část vychází ze studijního textu [9].

3.1.1 Rastrová grafika

Rastrová grafika je způsob popisu obrazové informace ve formě rastrové matice (mřížky, bitmapy), diskrétně. Jeden prvek matice se nazývá pixel. V tomto popisu uchováváme v rastrové matici pouze informaci o podobě zobrazených objektů, ne o objektech samotných. Jakmile je objekt jednou zobrazen, jako takový zaniká a uloženou informaci lze poté změnit pouze na úrovni jednotlivých pixelů. Informace je tedy uložena s konečnou úrovní detailu. Když se podíváme na detail dané matice, můžeme vidět až jednotlivé pixely, ale úroveň detailu zobrazované informace se nezmění.

3.1.2 Vektorová grafika

Vektorová grafika je způsob popisu obrazové informace ve formě skupiny vektorových entit, neboli primitiv (úseček, kružnic, křivek a polygonu), analyticky. Všechny objekty definované ve vektorové grafice jsou složeny ze seznamu těchto entit. Je tedy uchovávána informace o popisu zobrazených objektů a tak můžeme podle potřeby měnit jejich parametry a vlastnosti (polohu, velikost, barvu apod.). Jednotlivé entity jsou definovány přesně matematicky – úsečka svými koncovými body, kružnice středem a poloměrem apod. Nezávisí tedy na rozlišení jejich zobrazení, jako v rastrové grafice – zobrazení jejich detailu je vždy s maximální přesností.

3.1.3 Rasterizace

Většina současných zobrazovacích zařízení (monitory, projektory, ale i tiskárny) pracuje na principu rastrové grafiky – zajišťují zobrazení prostřednictvím množiny relativně malých bodů. Pro uložení v paměti počítače, zpracování, editaci a syntézu grafických dat je ale vhodnější vektorový popis.

Proto je třeba řešit proces *rasterizace* – převodu vektorové reprezentace dat na jejich rastrovou formu (mapu bodů). Tento proces je bez problému řešitelný pro všechny druhy vektorových entit. Probíhá automaticky a kvůli rychlosti zobrazení je snaha o jeho co nejvyšší optimalizaci. Proto je to jeden ze stěžejních prvků, který je hardwarově akcelerován pomocí grafických akcelerátorů.

Opačným procesem je vektorizace, která naopak není triviální a mnoha případech je nejednoznačná (vyžaduje manuálních korekcí).

3.2 Historie

Tato část čerpá z [27].

Historie akcelerované počítačové grafiky úzce souvisí s vývojem grafických akcelerátorů. Původně sloužily grafické procesory pouze jako digitálně-analogový převodník obrazových dat z videopaměti na zobrazovací zařízení (převážně monitor). První myšlenky přesunout výpočet často používaných grafických operací (rasterizaci apod.) na grafický čip se objevily v 70. letech 20. století. První grafický akcelerátor pro osobní počítače se objevil v 80. letech, ale jeho vysoká cena a malá kompatibilita bránila většímu rozšíření.

Na počátku 90. let byl představen první 2D akcelerátor na samostatném čipu. Následně byla 2D akcelerace integrována do grafických čipů všemi hlavními výrobci.

Dále se zvyšoval zájem o renderování (vykreslování) 3D grafiky v reálném čase – převážně díky videohrám. Jedny z prvních, masově vyráběných 3D grafických akcelerátorů, byly obsaženy právě v herních konzolích – *PlayStation* a *Nintendo 64*. Na osobní počítače zpočátku výkonné čipy obsahovaly pouze čistě 3D akcelerační funkce (zcela postrádaly funkce na akceleraci ve 2D) – příkladem budiž karta 3dfx Voodoo. Nicméně s vývojem výrobních technologií se začaly 2D akcelerace, 3D akcelerace a akcelerace videa integrovat všechny do jednoho čipu. Jedním z prvních zástupců byly čipsety *Verite* od firmy Rendition.

V 90. letech se také objevily jednotná grafická rozhraní *OpenGL* a *DirectX*, které byly jedny z hlavních hnacích sil vývoje hardwaru. Významným krokem v 3D renderování bylo představení hardwarové podpory transformace a osvětlení (T&L) – poprvé u karty *nVidia GeForce 256*.

V roce 2001 se objevily první programovatelné *shader* jednotky (poprvé na čipu *nVidia GeForce 3*). To znamená, že každý bod či vrchol může být před zobrazením ještě zpracován krátkým programem na grafickém procesoru. Rokem 2002 se přidala podpora smyček a matematických operací v shaderech (*ATI Radeon 9700*), čímž se GPU svou flexibilitou přiblížilo CPU. Díky tomu lze vytvářet mnoho pokročilých grafických efektů. Pro velkou výkonnost se v dnešní době grafické čipy také hojně využívají k obecným výpočtům (*General-Purpose computing on Graphics Processing Units*, zkratka GPGPU).

Díky velké podpoře hráčské komunity a stále zvyšujícími se nároky na kvalitu a rychlost zobrazení počítačové grafiky se 3D akcelerátory stále vyvíjí a neustále roste jejich výkon.

3.3 Grafické knihovny

Pro práci s akcelerovanou počítačovou grafikou je třeba využít speciálních grafických knihoven, které nám zpřístupní funkce grafického akcelerátoru. V dnešní době existují prakticky jen dvě hlavní, podporované a průběžně vylepšované grafické knihovny – OpenGL, vyvinutá firmou Silicon Graphics Inc., a DirectX, vyvinutá firmou Microsoft.

3.4 Knihovna OpenGL

OpenGL (*Open Graphics Library*) je průmyslový standard specifikující multiplatformní rozhraní pro tvorbu aplikací 2D a 3D počítačové grafiky. Je široce využívána v profesionálních grafických aplikacích, jako jsou CAD[1] systémy, vědeckotechnické vizualizace apod., ale také při tvorbě počítačových her. Knihovna je vyvíjena a spravována neziskovým konsorciem Khronos Group [24].

3.4.1 Historie

Ke konci 80. let 20. století vydala společnost Silicon Graphics, Inc. (SGI) rozhraní pro programování 3D aplikací s názvem IRIS GL. Jeho původním účelem byla podpora vývoje grafických aplikací, jako jsou CAD systémy a animace. IRIS GL umožňovala programátorovi na obrazovku vykreslovat 3D objekty, primitiva, a to pouze v jejich vlastním operačním systému IRIX. IRIS GL vytvářela celý okenní systém a spravovala data na obrazovce. Existovalo kolem 1 500 aplikací, které ji používaly pro vykreslování grafiky. Její nevýhoda byla hlavně v omezeních vyplývajících z používání vlastního, proprietálního okenního systému, který velmi ztěžoval její přenositelnost[18].

V roce 1992 vydala SGI nové 3D rozhraní s názvem Open Graphics Library, zkráceně OpenGL. Naproti původnímu IRIS GL systému byly v OpenGL odstraněny proprietální funkce a OpenGL tak umožňovalo vykreslovat 3D objekty nezávisle na použitém okenním systému. Také díky toho se z něj později stal průmyslový standard. První hra, která naplno využila možností tohoto rozhraní a hardware akcelerace, byl Quake od id Software (1996) [18]. Rozdíl mezi softwarovým vykreslováním a akcelerací s použitím OpenGL lze vidět na obrázku 3.1 (převzatého z [19]).

Podpora nových funkcí OpenGL probíhá pomocí přídávání tzv. *extensions* (rozšíření), které specifikují novou funkcionalitu. Vývoj těchto rozšíření byl spravován konsorciem ARB (*Architecture Review Board*), založeným roku 1992, dokud se nestal součástí konsorcia

[1] CAD (Computer Aided Design) je systém pro počítačovou podporu projektování.

Obrázek 3.1: Hra Quake (1996) bez (vlevo) a s využitím HW akcelerace (pomocí OpenGL).

Khronos Group (2006)[2]. Odtud také mají rozšíření zpravidla ve svém názvu zkratku ARB. Pokud GPU konkrétní rozšíření nepodporuje, může být emulováno softwarem za cenu horší výkonnosti.

V roce 2004 vyšla OpenGL verze 2.0, která přinesla podporů programovatelných jednotek (shaderů) – a to pomocí jazyku GLSL (*OpenGL Shading Language*).

Verze 3.0 (2008) přinesla první revizi API od počátku vývoje knihovny – mnoho funkcí bylo označeno jako zastaralých (například podpora maticových operací), ale byly zachovány pro zpětnou kompatibilitu.

Aktuální je verze 4.1 z roku 2010. Všechny verze OpenGL jsou zpětně kompatibilní, což znamená, že program napsaný ve staré verzi půjde spustit i na novější [24].

3.4.2 Struktura a způsob práce

OpenGL je procedurální knihovna založena na jazyce C, ale je použitelná téměř ze všech ostatních procedurálních programovacích jazyků. Práce s knihovnou spočívá ve volání více než 250 funkcí, které můžou být použity k vykreslení složité scény. Vše, co se vykresluje, je složeno pouze ze základních geometrických primitiv. Knihovna je založena principu stavového stroje, což znamená, že se nastaví nějaký stav (např. barva), a následně se všechno vykreslování provádí s tímto stavem. Výhodou je menší počet parametrů funkcí a fakt, že jedním příkazem lze změnit způsob vykreslení celé scény.

OpenGL využívá vykreslovací řetězec (*rendering pipeline*) – na vstupu jsou objekty 3D scény, definované souřadnicemi vrcholů, na výstupu pak 2D obrázek (rastr). Vykreslování probíhá do paměti snímku (*framebufferu*) [10].

[2] Více na http://www.opengl.org/about/arb/.

Při vytváření snímku je třeba postupnými transformacemi převést trojrozměrné souřadnice objektů na pozici pixelů obrazovky. Tranformace jsou v OpenGL reprezentovány násobením matic o velikosti 4×4. Matice jsou tři – *model-view* (nastavení kamery a umístění objektu), *projection* (volba projekce, viz kap. 4.3.1) a *viewport* (namapování na souřadnice okna) [10].

S verzí OpenGL 2.0 byl fixní vykreslovací řetězec nahrazen programovatelným – přibyla podpora vertex[3] a fragment[4] shaderů, od verze 3.2 pak geometry[5] shaderů. Podporované jazyky jsou GLSL a Cg (*C for Graphics*).

3.4.3 Výhody a nevýhody

Asi největší výhodou knihovny OpenGL je její přenositelnost na většinu počítačových platforem a nezávislost na hardwaru.

Nevýhodou může být, že v rámci maximální přenositelnosti, jako čistě grafická knihovna, nepodporuje platformě závislé funkce, jako je vytváření oken, zpracování událostí, vstupy z klávesnice a myši apod., takže při vývoji aplikací se buď musí volat funkce poskytované přímo operačním systémem, anebo se využije některé z multiplatformních rozšiřujících knihoven.

3.4.4 Rozšiřující knihovny

Pro OpenGL existují dvě nejrozšířenější přenositelné nadstavbové knihovny – GLUT a SDL.

GLUT (*OpenGL Utility Toolkit*) je knihovna pro snažší vývoj malých až středně velkých programů v OpenGL. Základem je platformě nezávislá práce s okny, zpracování událostí a práce s klávesnicí a myší. Dále podporuje základní nabídky (menu) a vykreslování různých složitějších útvarů. Hodí se zejména pro výuku, kdy už pomocí pár příkazů lze vytvořit OpenGL okno. Pro využití ve větších aplikacích se moc nehodí, jelikož obsahuje jen málo dalších funkcí. Omezením může být také nutnost použití *callback* funkce, ve které probíhá hlavní výpočet – problém v aplikacích, které vyžadují větší kontrolu nad zpracováním událostí. Knihovna už také neprochází dalším vývojem [7].

SDL (*Simple DirectMedia Layer*) je multiplatformní, multimediální knihovna navržená pro nízkoúrovňový přístup k audiu, vstupním zařízením (klávesnice, myš, joystick), 3D hardwaru (pomocí OpenGL) a 2D paměti snímku (framebufferu). Zastřešuje většinu funkcí operačních systémů (vytváření oken, zpracování událostí, časovače, podpora vláken apod.) a tím umožňuje téměř stoprocentní přenositelnost. Knihovna je jednoduchá (je to vlastně prostředník (*wrapper*) nad platformě závislými funkcemi operačního systému) a v základní verzi je relativně malá, další funkcionalita je dostupná s dalšími nadstavbami (podpora sítě, písem, obrázků apod.). Díky LGPL licenci a otevřenému zdrojovému kódu je využívána ve velkém množství aplikací, jako jsou multiplatformní hry či videopřehrávače [20].

Obě z knihoven v základu podporují jazyky C a C++, ale je možné využít i dalších.

[3] Vertex shader je program, který se provede na každém vrcholu vstupní geometrie.
[4] Fragment (pixel) shader je program, který se provede na každém pixelu na obrazovce.
[5] Geometry shader je program, který umožňuje přidávat a odebírat vrcholy a tím upravovat geometrii.

3.5 Knihovna DirectX

DirectX je sada knihoven poskytujících aplikační rozhraní pro tvorbu multimediálních aplikací, především počítačových her, na platformách firmy Microsoft. Kromě podpory akcelerované grafiky podporuje také přístup ke zvukovým zařízením, ovládacím prvkům, počítačové síti apod. Knihovna je hojně využívána v nejnovějších herních titulech.

3.5.1 Historie

Tato část čerpá z [26].

Vznik první verze DirectX souvisí s příchodem operačního systému Windows 95. Oproti dřívějšímu operačního systému MS-DOS, Windows 95 nepodporoval kvůli chráněnému paměťovému modelu přímý přístup k hardwaru – grafice, vstupním a zvukovým zařízením apod., takže práce se zařízeními byla kvůli mnoha vrstvám operačního systému pomalá. Proto herní vývojáři stále radši vyvíjeli pro MS-DOS a nebyl důvod přecházet na nový operační systém. To začal Microsoft rychle řešit a výsledkem se stalo řešení s názvem DirectX.

První verze byla uvolněna v září roku 1995 pod názvem *Windows Games SDK*. Druhá verze už se stala přímo součástí operačních systému Windows 95 OSR2 (*OEM Service Release*) a Windows NT 4.0 v polovině roku 1996. Microsoft poté začal s velkou propagací a počty titulů se na tuto platformu začaly rozšiřovat.

Z dalších verzí stojí za zmínku verze 7.0 uvedená v roce 1999, která přinesla podporu hardwarové transformace a osvětlení (T&L). Verze 8 (2000) přinesla podporu vertex a pixel shaderů. Verze 10 (2006) vyšla exkluzivně na Windows Vista a přinesla kompletní přepracování knihovny.

Aktuální (2011) je verze DirectX 11 z roku 2009, která obsahuje novinky jako podporu akcelerace fyziky či podporu obecných výpočtů na GPU (GPGPU). Verze DirectX jsou zpětně kompatibilní. Od verze 10, ve které se změnilo rozhraní, jsou starší verze pouze emulovány (pokles výkonu). Pro vyjádření pokročilosti GPU se často využívá označení verze DirectX, kterou plně podporuje.

3.5.2 Výhody a nevýhody

Výhodou knihovny DirectX je, že vývojáři poskytuje celou řadu modulů s podobným rozhraním potřebných k tvorbě multimediálních aplikací – 2D/3D grafika, zvuky, hudba, vstupy, síť a další. Tudíž není třeba využívat dalších externích knihoven. Dále, díky spolupráci Microsftu s výrobci grafických čipů, poskytuje přístup vždy k nejnovějším grafickým technologiím.

Nevýhodou je prakticky nulová přenositelnost na jiné platformy než Microsoft Windows[6].

[6] Kromě například implementace *WineHQ* v omezené formě na unixových systémech, viz http://www.winehq.org/.

Kapitola 4

Návrh

4.1 Koncepce hry

Základní koncepce hry je prakticky stejná jako u výše zmiňované hry Spacewar! – dva hráči, využívající jeden počítač, z nichž každý ovládá svou vesmírnou loď a na ploše jedné obrazovky (viděné shora) se vzájemným soubojem snaží zničit loď hráče druhého. Při zničení lodě má bod protihráč a souboj se opakuje. Cílem je nasbírat co nejvíce bodů.

Pohyb lodí ovládaných hráči je založen na newtonovské fyzice se setrvačností určenou směrem zážehu tryskového motoru. Tento způsob pohybu byl obsažen už ve hře Spacewar! a jejích následovníků, ale také v dalších – například u hry Asteroids[1]. Navíc je uvažováno samovolného zpomalování. Pokud loď, či jiný herní objekt pohybující se po hrací ploše, dosáhne kraje obrazovky, objeví se na kraji opačném.

Souboj mezi hráči probíhá pomocí různých zbraní, které se liší parametry jako je velikost poškození, kadence, či délka letu projektilu, aj. Kromě základního kanónu je možné využít i rakety a laser, které se dají dosáhnout přes systému bonusů. Pokud se lodě vzájemně srazí, jsou obě zničeny a výsledek kola je nerozhodný.

Každá loď má dvě vlastnosti – štíty a energii. Použití každé ze zbraní stojí nějakou část energie a zásah ze zbraně zase snižuje hodnotu štítů. Pokud je energie na nule, střelba není možná, pokud jsou na nule štíty, loď je zničena. Obě tyto vlastnosti mají pomalou regeneraci a dají se doplňovat přes systém bonusů. Existuje také možnost přesměrování energie do štítů a opačně, což se často může hodit.

Výše zmiňovaný systém bonusů funguje následovně. Na hrací ploše jsou v náhodných intervalech generovány na náhodných pozicích bonusy, které se aplikují na loď hráče, který přes bonus „proletí“ a tím jej sebere. Jak už bylo zmíněno, bonusy buď přidávají energii a štíty, anebo zbraně. Při sebrání bonusové zbraně se doplní energie na maximum a při jejím vyčerpání se zbraň přepne zpátky na kanón.

Pro větší zábavnost je po vzoru původních her možnost uprostřed hrací plochy zapnout planetu, která svou gravitací stahuje lodě a projektily k sobě. V případě nárazu do planety ja loď okamžitě zničena.

Celkový vzhled hry by měl dávat retrospektivní dojem, proto je grafika složena pouze z jednoduchých, drátěných primitiv. Díky použití barevných přechodů, průhlednosti, vyhlazování čar a částicových efektů lze i při zachování retro vzhledu dosáhnout pěkné a efektně vypadající grafiky (podobně jako u hry Geometry Wars, zmiňované v části 2.4.2).

Ze zvolených vlastností vyplývá, že návrh hry vyvíjené v této práci kombinuje různé

[1] Jednu z mnoha variací lze najít na http://www.play.vg/games/4-Asteroids.html.

vlastnosti z podobných her na toto téma probraných v části 2.4.1. Výběr kombinací těchto vlastností byl od začátku cílen především na dobrou a jednoduchou hratelnost a zároveň jednoduchý, ale efektní vzhled. Návrh všech výše zmiňovaných herních částí je rozebrán v následujících podkapitolách.

4.2 Platforma

Jako první je třeba zvolit platformu, na které bude hra vyvíjena. To znamená programovací jazyk, grafickou knihovnu, a případně další, doplňující knihovny.

4.2.1 Programovací jazyk

Pro programování her se nabízí jazyk C++. Pro tuto aplikaci je vhodný z několika důvodů:

- kompilovaný a tudíž relativně rychlý – pro hry důležité
- podpora většiny používaných grafických knihoven
- přenositelný na různé systémy
- dovoluje objektový návrh, jehož výhoda je ve znovupoužitelnosti a rozšiřitelnosti kódu

Pro tuto, ne až tak velkou aplikaci tyto výhody nebudou tak stěžejní, ale do budoucna, pro případná rozšíření, by se mohly jevit jako dobrý základ.

4.2.2 Grafická knihovna

Jako grafická knihovna byla pro podporu akcelerace vybrána knihovna OpenGL. Její výhody jsou následující:

- podpora většiny počítačových platforem (přenositelnost)
- je podporována drtivou většinou grafických karet
- jednoduchá práce, dobrá dokumentace a velká komunita vývojářů

V současnosti (2011) ja aktuální verze knihovny 4.1. Jelikož v naší aplikaci žádné z novinek z novějších verzí nepoužijeme, v rámci co největší hardwarové kompatibility se spokojíme s verzí 2.x. Novější verze jsou totiž zpětně kompatibilní, což se ale nedá říct o opaku.

4.2.3 Doplňující knihovny

Jelikož OpenGL je čistě grafická knihovna a nemá ostatní multimediální podporu, důležitou zejména k tvorbě her, bude v aplikaci navíc použita knihovna SDL. Její pro nás výhodné vlastnosti jsou zejména následující:

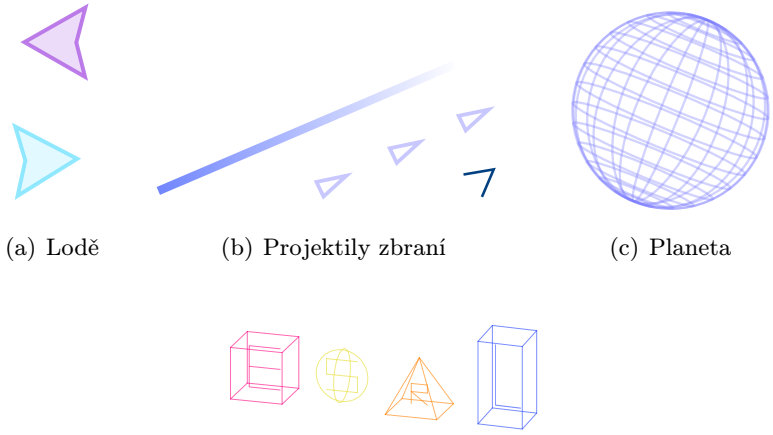
- podporuje OpenGL
- přenositelnost na velké množství platforem
- umožňuje správu oken a zpracování událostí
- podpora klávesnice a myši
- časem prověřená, hodně používaná a stále jsou ve vývoji nové verze

4.3 Vzhled

Hlavní požadavek na vzhled aplikace je prostý – retrospektivní, ale zároveň moderní. Obě tyto vlastnosti je možno šikovně skloubit, důkazem je v kapitole 2.4.2 zmiňovaná hra Geometry Wars.

Původní hry na téma vesmírného souboje se skládaly pouze z grafiky složené ze základních geometrických primitiv, a to ve dvou rozměrech a jen dvou barvách – bílé (objekty) a černé (pozadí). Na pozadí herní plochy se nacházely tečky znázorňující hvězdy. Jako základ retrospektivního vzhledu vyvíjené aplikace tedy také slouží pouze základní primitiva. Pro dosažení modernějšího stylu je ale použito více barev a u vybraných objektů také trojrozměrného zobrazení. Primitiva jsou vykreslovány s různou tloušťkou, průhledností a s využitím vyhlazování (*antialiasingu*) čar. Na pozadí herní plochy je navíc vyobrazena mřížka. Veškerá grafika je vykreslována za použití hardwarové akcelerace. Pro ještě lepší pocit ze hry jsou obsaženy různé efekty a animace, kterým se věnuje kapitola 4.7. Pomocí těchto vlastností by měl být požadavek na vzhled aplikace splněn. O vzhledu grafického uživatelského rozhraní pojednává kapitola 4.10.

Návrh vzhledu herních objektů je na obrázku 4.1. Lodě hráčů jsou vyobrazeny jako dvourozměrný polygon ve tvaru trojúhelníku s průhlednou výplní (4.1(a)). Na obrázku 4.1(b) jsou projektily zbraní – laser, kanón a raketa. Laser se skládá z čáry větší tloušťky, která má přechod od plné barvy až do zcela průhledné. Vyobrazením planety (4.1(c)) se dostáváme k trojrozměrným objektům – planeta jako drátěná koule. Bonusy (4.1(d)) jsou v pořadí energie, štíty, raketomet a laser. Každý bonus se pro snažší identifikaci skládá z jiného tvaru a uprostřed vloženého počátečního písmena.



(a) Lodě (b) Projektily zbraní (c) Planeta

(d) Bonusy

Obrázek 4.1: Návrh vzhledu herních objektů.

4.3.1 Projekce

Tato podkapitola čerpá ze studijního textu [9].

Po návrhu herních objektů je třeba vybrat způsob, jak objekty zobrazit na obrazovce. Protože většina zobrazovacích zařízení pracuje ve 2D, musí se 3D objekty redukovat na tuto dimenzi při zachování parametrů použitého zobrazení. Toho se dosahuje pomocí projekce,

která provádí redukci dimenze prostoru v definovaném směru (nejčastěji, a také v našem případě, ve směru osy $-z$). Rozlišujeme dva hlavní druhy projekce – paralelní a perspektivní:

Paralelní (rovnoběžná) projekce je lineární projekce, která zobrazuje vrcholy promítaných objektů prostřednictvím rovnoběžných paprsků. To znamená, že velikost objektů je nezávislá na jejích vzdálenosti od průmětny. Podle úhlu dopadu paprsků na průmětnu se rozlišuje ještě na kolmou (promítací paprsky jsou kolmé na průmětnu) a kosoúhlou (úhel dopadu paprsků na průmětnu je menší než 90 °).

Perspektivní (středová) projekce je nelineární projekce, která zobrazuje vrcholy promítaných objektů prostřednictvím paprsků protínajících se v jednom bodě, ve středu projekce (většinou pozice pozorovatele). To znamená, že velikost objektů je nepřímoúměrně závislá na jejich vzdálenosti od průmětny. Čím je objekt blíže průmětně, tím je jeho obraz větší a naopak. Tato projekce odpovídá lidskému vidění reálného světa.

V aplikaci je využito kolmé (ortogonální) paralelní projekce, protože ve scéně není použit pohyb objektů ve směru z-ové souřadnice (hloubky) a také pohled na herní plochu je statický (hra se odehrává na ploše jedné obrazovky a veškeré herní akce uvažují jen 2D prostor), takže je zbytečné řešit perspektivu. V OpenGL se ortogonální projekce nastavuje funkcí `glOrtho`, která si jako parametry bere rozměry pohledového objemu[2].

4.3.2 Průhlednost

V počítačové grafice se průhlednosti dá dosáhnout pomocí míchání barev (*blendingu*). Reprezentace barvy je, kromě hodnoty každé z její tří barevných složek (červená, zelená, modrá), obohacena ještě o hodnotu čtvrtou, tzv. alfa kanál, která udává velikost průhlednosti. To znamená, že pokud grafický objekt překrývá jiný objekt, barva objektu na pozadí bude v daném bodě zobrazena s intenzitou danou průhledností bodu v popředí. Při používání blendingu je důležité objekty vykreslovat ve správném pořadí.

Blending lze v OpenGL zapnout příkazem `glEnable(GL_BLEND)` a je třeba nastavit jeho správnou konfiguraci (míchací rovnici) pomocí funkce `glBlendFunc`. V aplikaci jsou použity parametry `GL_SRC_ALPHA` a `GL_ONE`[3].

4.3.3 Vyhlazování čar

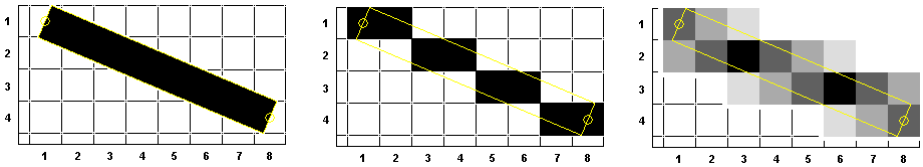
Vyhlazování (*antialiasing*) čar je technika pro odstranění zubatých hran vzniklých při rasterizaci. Čára je ve vektorové reprezentaci určena dvěma uzlovými body (počáteční a koncový). Pro zobrazení je ale nutné vektorovou reprezentaci převést na mapu bodů (obr. 4.2(a), převzat[4]). Protože body, které dokáže zobrazovací zařízení zobrazit, mají pouze omezenou hustotu, podaří se nám na ně namapovat čáru jen přibližně a vznikají zubaté hrany (obr. 4.2(b)).

Pokud v OpenGL zapneme vyhlazování čar (`glEnable(GL_LINE_SMOOTH)`, bude každému bodu rastrové mřížky, přes který čára aspoň částečně prochází, nastavena její původní barva s vypočtenou hodnotou alfa kanálu. Výpočet hodnoty alfa kanálu závisí na vzdálenosti

[2] Pohledový objem je část 3D prostoru, která obsahuje viditelné objekty. Při paralelní projekci je ohraničen okraji okna protaženými do omezené hloubky ve tvaru hranolu. U perspektivní projekce je využito tvaru komolého jehlanu.

[3] Více infomací a význam parametrů na http://www.root.cz/clanky/opengl-28-blending/.

[4] Obrázky převzaty z http://courses.engr.illinois.edu/ece390/archive/archive-f2000/mp/mp4/anti.html.



(a) Vektorová reprezentace (b) Rastr. rep. bez vyhlazení (c) Rastr. rep. s vyhlazením

Obrázek 4.2: Vyhlazování (antialiasing) čar.

středu bodu od středu čáry. Čím je vzdálenost menší, tím je bod méně průhledný. Touto metodou se opticky docílí vyhlazení (obr. 4.2(c)). Pomocí funkce `glHint(GL_LINE_SMOOTH, GL_NICEST)` lze ovlivnit přesnost výpočtu a tím kvalitu zobrazení (za cenu vyšší náročnosti) [2]. Protože se využívá alfa kanálů, je třeba mít v OpenGL zapnutý výše zmíněný blending.

4.4 Pohyb objektů

Velmi důležitou částí je rozpohybování herních objektů, které tvoří jádro hry. Pohybem je myšlena jak změna jejich pozice v čase, tak i změna rotace či měřítka (například u efektů).

4.4.1 Geometrické transformace

Tato podkapitola čerpá ze studijního textu [9].

Geometrické transformace jsou jedním z nejdůležitějších a nejpoužívanějších nástrojů moderní počítačové grafiky. Lze je chápat jako změnu pozice vrcholů vektorových objektů v aktuálním souřadnicovém systému, anebo jako změnu souřadnicového systému. Umožňují nám provádět operace posunutí, otočení, změnu měřítka či zkosení těchto objektů (jejich uzlových bodů, vrcholů, kterými jsou popsány). Tyto operace jsou základ pro vytváření pohybů či animací.

Aby se s těmito operacemi dalo pracovat jednotně, je třeba všechny tyto operace převést na jednotnou formu. Řešením jsou homogenní souřadnice bodu. Pro jejich reprezentaci se využívá maticového zápisu. Rozměry matice jsou v 3D kartézském systému 4×4, ve 2D pak 3×3.

Aktuální stav každého objektu (jeho pozice, rotace a měřítko) je pak určen jednou transformační maticí, která umí vyjadřit jakoukoli obecnou transformaci – získá se násobením dílčích základních transformačních matic, které jsou uvedeny níže. Toto násobení matic musí být z pravé strany a v přesném pořadí provádění jednotlivých transformací.

Jednotlivé operace jsou definovány následovně:

Posunutí bodu v trojrozměrném prostoru o vektor posunutí $T(d_x, d_y, d_z)$ je definováno transformační maticí 4.1.

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ d_x & d_y & d_z & 1 \end{bmatrix} \tag{4.1}$$

V OpenGL lze toto posunutí realizovat pomocí funkce `glTranslate(dx, dy, dz)`, která současnou matici vynásobí maticí posunutí [13].

Otočení bodu v trojrozměrném prostoru o úhel α se středem v počátku souřadného systému je definováno třemi různými transformačními maticemi R_x, R_y, R_z (vztahy 4.2 – 4.4), každá pro rotaci kolem jedné ze tří souřadných os (X, Y a Z).

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

$$R_y = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & \cos\alpha & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

$$R_z = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

V OpenGL lze tyto změny rotací kolem všech tří os realizovat jednotnou funkcí `glRotate(`α`, x, y, z)`, která současnou matici vynásobí rotační maticí rotovanou kolem vektoru $\vec{v}(x, y, z)$ [13].

Změna měřítka v trojrozměrném prostoru o faktory změny měřítka S_x, S_y a S_z je definována transformační maticí 4.5.

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

Pokud je faktor změny měřítka $S_{x,y,z} > 1$, dochází ke zvětšení, pokud je $0 < S_{x,y,z} < 1$, dochází je zmenšení a pokud je $S_{x,y,z} < 0$, dochází k převrácení (zrcadlení). V OpenGL lze tuto změnu měřítka realizovat funkcí `glScale(`S_x, S_y, S_z`)`, která současnou matici vynásobí maticí změny měřítka [13].

4.4.2 Rychlost a směr

Rychlost a směr pohybu každého objektu, který mění svou pozici, je v této aplikaci reprezentován jedním dvourozměrným vektorem (pohyb po ose z není využíván). Pokud bude objekt opakovaně posouván o tento vektor (bude přičítán k aktuální pozici objektu), bude se ve směru vektoru pohybovat, a to rychlostí, která odpovídá délce tohoto vektoru a frekvenci posouvání.

Kvůli akcelerace lodí, jejich samovolnému zpomalování a působení gravitace je třeba zavést druhý vektor, který bude udávat změnu rychlosti. Tento vektor bude poté k aktuální hodnotě vektoru rychlosti přičítán, a tím ji bude měnit. Pokud bude jeho délka nulová, rychlost bude beze změny. Pokud bude kladná, aktuální rychlost bude v jeho směru ovlivňována (směr a délka výsledné rychlosti bude záviset na součtu těchto vektorů)[5].

[5] Chování na základě Newtonových pohybových zákonů.

V aplikaci je potřeba směr vektoru určovat na základě hodnoty úhlu – například pro akceleraci lodě ve směru aktuálního natočení. Vytvoření takového vektoru $\vec{v}$ se směrem určeným úhlem α je znázorněno pomocí goniometrických funkcí ve vztahu 4.6.

$$\vec{v} = (\cos\alpha, \sin\alpha) \tag{4.6}$$

Nastavení délky vektoru $\vec{v}$ se poté provede jeho normalizací (získáme jednotkový vektor udávající pouze směr) a následným vynásobením skalární hodnotou s udávající jeho délku v daném směru (vztah 4.7).

$$\vec{v} = \frac{\vec{v}}{|\vec{v}|} \cdot s \tag{4.7}$$

Operace $|\vec{v}|$ (délka vektoru $\vec{v}$) je definována vztahem 4.8, ve kterém x a y představují jednotlivé složky vektoru.

$$|\vec{v}| = \sqrt{x^2 + y^2} \tag{4.8}$$

4.4.3 Stabilita rychlosti pohybu

Protože u počítačových her dochází prakticky nepřetržitě ke spoustám změn stavů, například pohybů herních objektů, animacím apod., je nutné stav hry také nepřetržitě aktualizovat. Tímto přístupem se počítačové hry liší například od kancelářských aplikací, kde se aktualizace provádí jednou za čas např. po stisku nějakého tlačítka apod.

Rychlost aktualizací hry se často udává v hodnotě počtu snímků za vteřinu (*Frames Per Second*, FPS). Protože rychlost aktualizace závísí na výkonu počítače a není předem známo, s jakou frekvencí bude probíhat, nastává problém, jak pohybovat s herními prvky, aby byla rychlost pohybu na všech počítačích stejná (aby se za jednotku času vždy pohly o stejnou hodnotu). Pokud totiž v každé aktualizaci posuneme nějaký objekt o konstantní hodnotu, například 10 jednotek, při 30 FPS se za vteřinu posune o 300 jednotek, při 60 FPS ale už o 600 jednotek, čemuž chceme zabránit.

Tento problém se dá řešit například těmito způsoby:

- omezení maximálního počtu snímků za sekundu na konstantní hodnotu
- pohyb pouze v určených časových intervalech (obdoba časovačů z „klikacích“ aplikací)
- konstanty veškerého pohybu ovlinit o hodnotu rozdílového času mezi aktuálním a předešlým snímkem

V prvním případě se s aktualizací čeká, dokud neuplyne časová doba odpovídající požadovanému počtu snímků za sekundu a až poté se objekty aktualizují. Při tomto řešení nastává problém, když je počítač moc pomalý a nezvládá hru v požadovaném FPS. Ta pak běži zpomaleně.

Pohyb na základě časových intervalů ja podobný jako předchozí případ, jen si pro každý objekt můžeme vytvořit samostané časovaní, a každý z nich se poté aktualizuje v jiných intervalech. V předchozím případě se např. různá rychlost pohybu řešila různou velikostí přičítané konstanty, v tomto případě lze rychlost ovlivňovat také volbou intervalu. Problém tohoto řešení může být v „trhavém“ pohybu, protože objekt se při větších intervalech neauktalizuje plynule (stejně tak u předchozího řešení při nastavení moc nízké hodnoty FPS). Problém s nízkým FPS zůstává.

Poslední případ je použit v této aplikaci. V každém snímku se počítá rozdílový čas mezi aktuálním časem a časem předešlé aktualizace. Touto hodnotou jsou pak násobeny veškeré konstanty, které ovlivňují pohyb objektu (např. hodnota udávající rychlost apod.). Při nízkých FPS je rozdílový čas větší, a objekt se v jedné aktualizaci posune více, při vyšších FPS opačně. Čim je vyšší hodnota FPS, tím se objekt pohybuje plynuleji. Toto řešení se chová přirozeně a je jednoduché. Nevýhoda může být v tom, že je třeba na všech místech programu, kde se takový pohyb provádí, znát rozdílovou hodnotu mezi snímky a všechny konstanty jí násobit.

Rychlost pohybu objektů je tedy v této aplikaci nastavována pouze na základě velikosti přičítaného vektoru k aktuální pozici objektu. Frekvence přičítání se neuvažuje, protože ta je vždy maximální (každý snímek). Stabilita je zajištěna násobením přičítaného vektoru rozdílovým časem mezi aktuálním a předchozím snímkem.

4.4.4 Pohyb lodí

U pohybu lodí se uvažují čtyři vlastnosti: velikost a směr akcelerace, velikost samovolného zpomalování, působení gravitace a omezení maximální rychlosti.

Pro první tři případy stačí pracovat pouze s vektorem udávajícím změnu rychlosti, který je přičítán k její aktuální hodnotě. U čtvrtého případu je pak potřeba omezit velikost vektoru udávajícího rychlost.

Loď je na začátku v klidovém stavu – vektory rychlosti a její změny jsou v nulové hodnotě. Pokud jsou zažehnuty trysky, ve směru natočení lodě se nastaví vektor udávající změnu rychlosti do délky určené velikostí akcelerace. Tento vektor je následně přičten k vektoru rychlosti a loď se začne pohybovat (vektor rychlosti je k pozici lodě každý snímek přičítán). Další akcelerace probíhají stejně, výsledná rychlost je určována vždy součtem vektorů aktuální rychlosti a její nastavené změny v konkrétním směru.

Samovolné zpomalování je řešeno odčítáním vektoru udávajícího aktuální rychlost od vektoru udávajicího aktuální změnu rychlosti. Odčítaný vektor je ale násoben konstantou udávající velikost zpomalení. Tato konstanta by měla být v hodnotách menších než 1, aby bylo zpomalení vždy menší než aktuální rychlost. Tímto řešením se dosáhne stavu, že čím se objekt pohybuje rychleji, tím více je zpomalován.

Omezení maximální rychlosti na zadanou hodnotu se provede následovně: po přičtení změny rychlosti se zjistí velikost výsledné rychlosti (délka vektoru), a pokud je větší než zadaná hodnota, vektor se nastaví na její délku (normalizace a následné vynásobení hodnotou maximální rychlosti).

4.4.5 Působení gravitace

Planeta působí na zvolené objekty (lodě a projektily) určitou silou gravitace. To znamená, že je samovolně přitahuje k sobě. Čím je objekt blíže k planetě, tím je působení gravitace silnější.

Této vlastnosti je dosaženo ovlivňováním aktuální hodnoty změny rychlosti $\vec{a}$ objektu o vektor ve směru ke středu působení gravitace. Vektor $\vec{d}$ udávající směr a vzdálenost do středu gravitace se získá na základě aktuální vzájemné pozice planety ($P[x_p, y_p]$) a pozice objektu ($O[x_o, y_o]$) (vztah 4.9).

$$\vec{d} = (x_o - x_p, y_o - y_p) \tag{4.9}$$

Následně se spočítá hodnota z, udávající velikost změny rychlosti směrem k středu gravitace, jako podíl síly gravitace planety s a délky vektoru $\vec{d}$ (vztah 4.10).

$$z = \frac{s}{|\vec{d}|} \tag{4.10}$$

Tímto se dosáhne nepřímé úměrnosti – čím je délka vektoru $\vec{d}$ mezi planetou a objektem menší, tím je velikost změny rychlosti z objektu směrem ke středu gravitace větší.

Nakonec je od aktuální hodnoty změny rychlosti $\vec{a}$ objektu odečten vektor o délce udávající velikost změny rychlosti směrem ke středu gravitace. Získá se normalizací vektoru $\vec{d}$ (směr do středu gravitace) a jeho vynásobením hodnotou z (vztah 4.11).

$$\vec{a} = \vec{a} - \frac{\vec{d}}{|\vec{d}|} \cdot z \tag{4.11}$$

Síla gravitace planety s tedy udává, jak moc k ní bude objekt přitahován. Z řešení vyplýva, že pokud zadáme tuto hodnotu jako zápornou, planeta bude místo přitahování objekt odtahovat. Hodnotu lze také pro každý objekt měnit za pomoci násobitele, kterým bude její základní velikost vynásobena.

4.5 Zbraně a systém bonusů

Zbraně a systém bonusů tvoří základní hratelnost hry. Pomocí zbraní je prováděn nezbytný souboj mezi hráči a bonusy dále rozšiřují herní možnosti. Oba prvky úzce souvisí s lodními vlastnostmi – štíty a energií.

4.5.1 Vlastnosti lodí – štíty a energie

Každá loď je závislá na dvou vlastnostech – štítech, které ji chrání před vnějšími zásahy ze zbraní lodě protihráče a energii, která je naopak potřebná k používání zbraní vlastních.

Pokud hodnota štítů klesne na nulu, loď je zničena. Pokud je energie na nule, není možno využívat lodních zbraní. Obě tyto vlastnosti mají v pravidelných časových intervalech určitou hodnotu regenerace. Pro více možnosti taktizování existuje volba přesměrování energie do štítů a naopak.

Hodnota šítů i energie je reprezentována jako číselná hodnota v rozsahu 0–100. Na začátku kola mají oba hráči tyto hodnoty nastavené na maximum.

4.5.2 Zbraně

Zbraně jsou celkem tři a jejich vlastnosti jsou určeny chováním jejich projektilů. Vzhled projektilů je znázorněn na obrázku 4.1(b) na straně 20. Typy zbraní a vlastností jejich projektilů (včetně obsahu částicových efektů, viz 4.8) shrnuje tabulka 4.1.

Kanón je základní zbraň, která je nenáročná na energii a dává malé poškození. Její kadence je ale ze všech zbraní nejvyšší.

Raketa je silnější zbraní, má velký dolet a dává citelné poškození. Spotřeba energie vyjde až na pět střel. Projektil je obohacen jako jedinný o částicový efekt letu.

Vlastnost	Kanón	Raketa	Laser
velikost poškození (0–100)	5	30	90
cena energie (0–100)	2,5	20	50
doba života [ms]	650	1 250	50
kadence [ms]	100	800	1 000
rychlost	0.75	0.75	0
násobitel gravitace	20	20	0
částicový efekt trysek	ne	ano	ne
částicový efekt výbuchu	ano	ano	ne

Tabulka 4.1: Vlastnosti projektilů jednotlivých zbraní.

Laser se od předchozích zbraní liší. Obsahuje jeden delší projektil, který se objeví jen na velmi krátkou dobu a s nulovou rychlostí. Proto je to zbraň na blízko. Má velmi malou kadenci a velkou spotřebu energie, ale když už se zásah podaří, je často fatální, protože dokáže vzít skoro celou protivníkovu energii.

Zbraně fungují na principu projektilu, který je po vystřelení ze zbraně vytvořen, jsou mu předány atributy a následně se podle nich autonomě chová. To znamená, že se pohybuje zadanou rychlostí ve směru natočení lodě při výstřelu, dokud neuplyne doba jeho života. Při zásahu (detekce kolizí viz kap. 4.6) do nepřátelské lodě je lodi snížena hodnota štítů o velikost poškození projektilu a následně je projektil odstraněn, včetně případného vytvoření částicového efektu výbuchu (kap. 4.8.3).

Na projektily působí gravitace planety, její síla může být každému typu projektilu měněna pomocí násobitele. Pokud je nulový, gravitace na něj nemá vliv.

V základu má hráč k dispozici pouze kanón, ostatní zbraně lze dosáhnout přes systém bonusů.

4.5.3 Systém bonusů

Bonusy se dělí na dva hlavní typy – doplnění lodních vlastností (energie či štítů) a aktivace bonusové zbraně (rakety či laseru).

Bonusy jsou na hrací ploše generovány na základě dvou parametrů. Prvním z nich rozsah hodnoty generované generátorem pseudonáhodných čísel s rovnoměrným rozložením, která udává časový interval, za jaký se má vygenerovat další bonus. Druhý parametr určuje, kolik může být na hrací ploše v jednu dobu maximálně bonusů. To znamená, že když se počet bonusů rovná maximální hodnotě, další už se negeneruje, dokud se počet bonusů nesníží.

Vzhled jednotlivých bonusů vygenerovaných na hrací ploše je znázorněn na obrázku 4.1(d) na straně 20. Bonusy se na hrací ploše animují a při jejich sebrání lodí (detekce kolize) je bonus na loď aplikován, je vytvořen efekt jeho sebrání (viz kap. 4.7) a nakonec je bonus odstraněn.

Popis typů bonusů (jejich akcí po aplikaci na hráčovu loď) je následující:

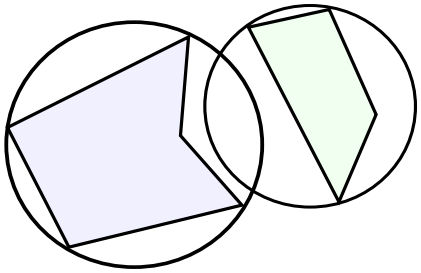
Doplnění lodních vlastností spočívá v tom, že je po sebrání bonusu jednoduše inkrementována hodnota představující konkrétní vlastnost (štíty o 35 jednotek, energie o 40). Pokud vlastnost přesáhne maximum (100 jednotek), je na maximální hodnotu upravena.

Aktivace bonusové zbraně funguje tak, že po sebrání bonusu je zbraň hráči nastavena jako aktuální a energie je doplněna na maximální hodnotu. Pokud hráč vyčerpá energii (že už nestačí na další střelbu ze zbraně), při následující střelbě je aktivován opět kanón a zbraň je odstraněna (za doplnění malé hodnoty energie). To znamená, že hráč může s bonusouvou zbrání sbírat bonusy pro doplnění energie a tak si prodloužit její životnost.

4.6 Kolize mezi objekty

Aby bylo možné ve hře zjistit a reagovat na zásahy ze zbraní, sebrání bonusu či náraz do planety, je třeba se věnovat detekci kolizí mezi těmito objekty. Pokud chceme v počítačových hrách detekovat kolize s velkou přesností, u více členitých objektů je to složitý a výpočetně náročný problém. Protože hry musí běžet v reálném čase, používá se mnoho zjednodušení a optimalizací [6].

Základní princip spočívá v obalení objektu do jiného, jednoduššího, u kterého je poté spočítání kolize s jiným, stejně obaleným objektem méně náročné. Objekty se nejčastěji obalují do kružnic (2D), koulí nebo *bounding boxu* („krabice“). Při těchto řešeních může nastat problém, který je znázorněn na obrázku 4.3 – objekt přesně nevyplní plochu obalovaného objektu a kolize je pak špatně detekována.



Obrázek 4.3: Nesprávná detekce kolize při průniku dvou kružnic.

Proto existují rozšíření těchto metod, kdy se objekt neobalí jen do jednoho obalového objektu, ale do jejich většího počtu (sférické nebo kubické rozdělení objektu, více v [11]).

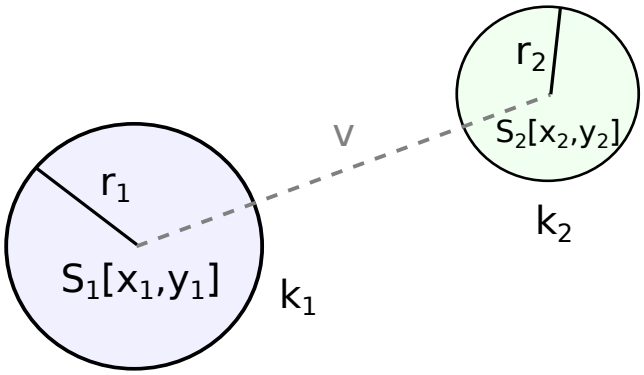
Jednoduché objekty, jakými jsou například úsečky, body nebo roviny se většinou neobalují, protože s nimi lze, na základě analytické geometrie, zjistit průnik přímo.

Další optimalizací je nekontrolovat kolize zbytečně u objektů, s kterými v aktuálním okamžiku vůbec nemůže nastat. Například, pokud se pohybujeme s objektem po herním světě, složeným z mnoha dalších objektů, je zbytečné kontrolovat kolize s objekty, které jsou velmi vzdáleny. Pro tyto účely se používá dělení prostoru do stromové hierarchie (binární rozdělování prostoru, oktalový strom). Více v [6].

V této aplikaci jsou použity metody průniku dvou kružnic a průniku kružnice a přímky. Dělení prostoru není použito, protože se v aplikaci nevyskytuje velký počet objektů.

4.6.1 Průnik dvou kružnic

Pro detekci kolize mezi loděmi, planetou, bonusy a projektily se používá detekce průniku dvou kružnic. Všechny tyto objekty mají určenou pozici a poloměr, ve kterém jsou vykresleny. Protože objekty kužnici vždy z větší části vyplní, problém z obrázku 4.3 nenastává. Objekty sice nejsou obaleny úplně přesně, ale to lze v rámci hratelnosti zanedbat.



Obrázek 4.4: Poloha dvou kružnic.

Poloha dvou kružnic je znázorněna na obrázku 4.4. Jejich průnik se vypočte s využitím analytické geometrie, konkrétně na základě vzájemné polohy dvou kružnic a vzdálenosti dvou bodů – velikosti vektoru.

Vektor $\vec{v}$ mezi středy obou křužnic S_1 a S_2 se získá pomocí vztahu 4.12.

$$\vec{v} = (x_2 - x_1, y_2 - y_1) \tag{4.12}$$

Jeho velikost udává vzdálenost mezi středy kružnic. Poloha dvou kružnic s poloměry r_1 a r_2 je poté následující:

$|\vec{v}| > r_1 + r_2$ – kružnice nemají společný bod, nedošlo ke kolizi

$|\vec{v}| = r_1 + r_2$ – kružnice mají jeden společný bod, došlo ke kolizi

$|\vec{v}| < r_1 + r_2$ – kružnice mají dva společné body, došlo ke kolizi

4.6.2 Průnik kružnice a úsečky

Tato metoda se používá pro detekci kolize lodě a projektilu laseru. Projektil je totiž reprezentován delší úsečkou a kdyby byl obalen do kružnice, docházelo by k problému z obrázku 4.3. Pro detekci kolize se tedy bude počítat průnik mezi kružnicí a úsečkou, a to na základě jejich vzájemné polohy [4] (viz analytická geometrie).

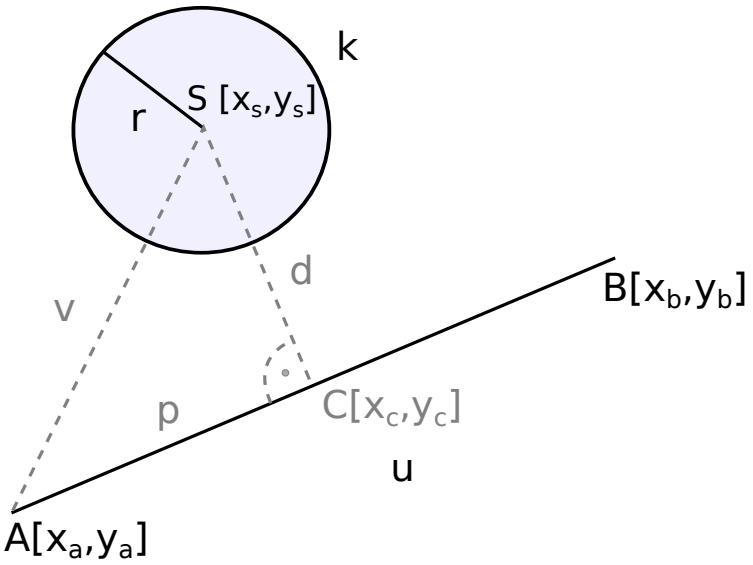
Poloha kružnice k, se středem v bodě S, a úsečky, určené krajními body A a B, je znázorněna na obrázku 4.5. Cílem je zjistit velikost vektoru $\vec{d}$ mezi středem kružnice a nejbližším bodem na úsečce, C.

Nejprve se spočte vektor úsečky $\vec{u}$ a vektor $\vec{v}$ mezi začátkem úsečky a středem kružnice (vztahy 4.13 a 4.14).

$$\vec{u} = (x_b - x_a, y_b - y_a) \tag{4.13}$$
$$\vec{v} = (x_s - x_a, y_s - y_a) \tag{4.14}$$

Dalším krokem je najít na úsečce nebližší bod (C) ke středu kružnice S. Abychom tohoto dosáhli, je třeba provést projekci vektoru $\vec{v}$ do vektoru $\vec{u}$. Toho se dosáhne skalárním součinem mezi vektorem $\vec{v}$ a jednotkovým vektorem cíle projekce ($\vec{u}$) (vztah 4.15).

$$|\vec{p}| = \vec{v} \cdot \frac{\vec{u}}{|\vec{u}|} \tag{4.15}$$



Obrázek 4.5: Poloha kružnice a úsečky.

Výsledkem je délka vektoru $\vec{p}$. Pokud touto délkou vynásobíme normalizovaný vektor $\vec{u}$, získáme hodnotu tohoto vektoru (vztah 4.16).

$$\vec{p} = |\vec{p}| \cdot \frac{\vec{u}}{|\vec{u}|} \tag{4.16}$$

Souřadnice nejbližšího bodu C se potom získají přičtením vektoru $\vec{p}$ k bodu udávajícímu začátek úsečky (vztah 4.17). V této fázi je třeba uvažovat speciální případ, a to krajní body úsečky. Pokud je $|\vec{p}| < 0$, pak nejbližší bod úsečky je na jejím počátku ($C = A$). Pokud je $|\vec{p}| > |\vec{u}|$, pak je nejbližší bod úsečky na jejím konci ($C = B$).

$$C = A \ + \vec{p} \tag{4.17}$$

Nakonec se vypočte vektor $\vec{d}$ mezi středem kružnice a nejbližším bodem na úsečce C (vztah 4.18).

$$\vec{d} = (x_s - x_c, y_s - y_s) \tag{4.18}$$

Velikost vektoru $\vec{d}$ pak udává vzdálenost středu kružnice od nejbližšího bodu úsečky. Poloha mezi kružnící a úsečkou je potom následující:

$|\vec{d}| > r$ – žádný společný bod, nedošlo ke kolizi

$|\vec{d}| = r$ – jeden společný bod (tečna), došlo ke kolizi

$|\vec{d}| < r$ – dva společné body (sečna), došlo ke kolizi

4.7 Efekty a animace

Efekty a animace jsou velmi důležitou součástí počítačových her. Často neovlivňují přímo hratelnost, ale dokáží značně vylepšit dojem ze hry. Vzhled hry obohacený o efekty také

zpravidla dává hře větší šanci na úspěch, protože hodně uživatelů se o tom, zda hru vůbec zkusí, rozhoduje na základě obrázků z ní.

V této aplikaci je použito několik jednoduchých animací a efektů. Efekty výbuchu a trysek jsou implementovány částicovým systémem, viz kapitola 4.8.

4.7.1 Animace

Animace jsou použity celkem dvě:

Rotace objektu kolem některé z jeho os – použito pro animaci planety s aktivní gravitací. K aktuální rotaci objektu je nepřetržitě přičítán určitý přírůstek v závislosti na čase mezi snímky. U planety je rotace prováděna kolem osy y.

Rotace objektu včetně jeho „levitace“ – použito pro animace bonusů na hrací ploše. Rotace stějně jako předchozí případ (také osa y), navíc je objekt posouván od své originální pozice v ose x o hodnotu funkce $\sin x$, kde x je inkrementovaná hodnota. Protože má funkce sinus harmonický průběh, objekt se bude plynule pohybovat nahoru a dolů a tím vznikne dojem levitace.

Animace se provádí nepřetržitě, dokud není vnějším vlivem ukončena (sebrání bonusu či vypnutí gravitace).

4.7.2 Efekty

Efekty se provádí jednorázově, jako reakce na nějakou akci. Délka jejich trvání je určena časovým intervalem. Typy efektů jsou následující:

Plynulý barevný přechod z jedné barvy do druhé – v určeném časovém intervalu t_l se lineárně interpoluje každá složka barvy[6] objektu c z počáteční hodnoty barvy c_s do koncové hodnoty c_e. Pomocí vztahu 4.19 se zjistí procentuální hodnota doby aktivity efektu p na základě aktuálního času t_c a koncového času efektu t_e. Tato hodnota se pak použije ve vztahu 4.20 jako váha interpolace. Výsledkem je interpolovaná hodnota barvy c v aktuálním čase efektu.

$$p = (t_e - t_c)/t_l \tag{4.19}$$

$$c = c_s + (c_e - c_s) \cdot p \tag{4.20}$$

Tento efekt je využit při zobrazování textu o stavu hry (viz kap. 4.10.1), při sebrání bonusu (viz níže) a u částicových efektů (kap. 4.8).

Plynulé zvětšování objektu s přechodem do ztracena – efekt při sebrání bonusu. Bonus se začne zvětšovat (měnit měřítko) a jeho barva je interpolována na koncovou barvu s nulovou hodnotou alfa kanálu – objekt plynule zmizí. V efektu je také obsažen text s názvem bonusu, který se žačne pohybovat směrem vzhůru (podél osy x) a jeho barva je také interpolována do ztracena.

[6] Barva je reprezentována jako čtveřice hodnot (červená, zelená, modrá, alfa kanál) s plovoucí řádovou čárkou v rozsahu 0 (nejnižší intenzita) až 1 (nejvyšší intenzita).

4.8 Částicové systémy

Za použití částicových systémů lze v počítačové grafice dosáhnout velkého množství pokročilých efektů – od výbuchů, přes různé fontány, až po animaci ohně.

Částicový systém je kolekce většího počtu jednotlivých elementů – částic. Každá částice má své atributy, jako je rychlost, barva, doba života apod. Chování každé z částic je autonomní – není závislé na částicích ostatních. Částice mají v rámci jednoho částicového systému zpravidla stejnou množinu atributů, lišících se pouze svými hodnotami. Společně pak částice vytváři určítý efekt [8].

Vzhled částice může být reprezentován například jen jako bod nebo úsečka, nejčastěji však jako sprite (např. pro animace ohně). Je ale možno využít jakékoli jiné reprezentace. Velmi často se využívá průhlednosti [8].

V této práci je použit částicový systém vytvářející efekty výbuchu a trysek.

4.8.1 Částice

Častice je v našem případě reprezentována pouze jednoduchou úsečkou zadané barvy, délky a tloušťky.

Částice je určena těmito parametry:

- pozice
- rychlost (udávající také směr)
- doba života
- počáteční a koncová barva
- úhel natočení – nastavován podle směru pohybu
- délka, tloušťka a zda zkracovat

Částice během svého života interpoluje svou barvu z počáteční do koncové (viz kap. 4.7.2) a pokud je zadáno zkracování, délka její úsečky se plynule zmenšuje (její délka je násobena procentuální hodnotou doby života částice – vztahem 4.19 z kapitoly 4.7.2).

4.8.2 Generátor částic

Generátor částic (*particle emitter*) má na starost vytváření částic. Generuje je v zadaném počtu a podle různých počátačních parametrů. Tyto parametry jsou pak předány nově vytvořené částici a ta se podle nich již autonomě chová, nezávisle na generátoru.

Počáteční parametry částic zpravidla nejsou určeny pevnými hodnotami, ale generují se na základě stochastických jevů s určeným rozsahem. Tím se dosáhne přirozenějšího dojmu, protože většina modelovaných efektů není dána pevnými hodnotami (např. výbuchy či hoření ohně).

4.8.3 Částicové efekty

Částicové efekty jsou vytvořeny dva a liší se především směrem generování částic.

Efekt výbuchu je použit pro znázornění zničení lodí, raket a projektilů kanónu. Částice v tomto efektu je třeba generovat do všech stran (celých 360 °). Použije se tedy generátoru pseudonáhodných čísel (s plovoucí řádovou čárkou) s rovnoměrným rozložením v rozsahu -1 až 1 a pro každou částici se vygenerují dvě hodnoty, x a y. Na základě těchto hodnot je poté vytvořen vektor, který udává náhodný směr v rozsahu celých 360 °. Tomuto vektoru se poté nastaví délka, udávající rychlost (viz kap. 4.4.2). Velikost rychlosti je určena také generátorem pseudonáhodných čísel s rovnoměrným rozložením, aby při výbuchu neletěly všechny částice ve stejné linii. Při tomto efektu je zapnuto zkracování čar a konečná barva není nastavena na průhlednou.

Efekt trysek je použit pro znázorění pohonu lodí a rakety. Efekt je tvořen stejně jako v předchozím případě, jen je omezen rozsah směru generování částic. Hodnota vektoru na pozici osy x je napevno nastavena na hodnotu -1, takže generování bude vždy probíhat jen do druhého a třetího kvadrantu. Rozsah na ose y je omezen na hodnoty od -0.25 do 0.25, kterými se dosáhne úžšího rozptylu. Doba života částic je generována také s využitím rovnoměrného stochastického jevu. Konečná barva je nastavena na zcela průhlednou, efekt končí ve ztracenu.

4.9 Ovládání pro dva hráče

Ovládání hry je umožněno pomocí klávesnice a myši. Aby styl ovládání vyhovoval každému uživateli, není vhodné jej nastavovat napevno. U hry pro dva hráče má toto ještě větší váhu. Proto je umožněno každému hráči si pomocí menu nastavit (kap. 4.10.3), jak hru ovládat.

Ovládání každé akce může být ve dvou stylech: buď je reagováno pouze na stisk tlačítka, nebo i na jeho držení. Při reakci pouze na stisk tlačítka se provede přiřazená akce jen jednou, a pro její opakování je nutné tlačítko stisknout znovu. V druhém případě se akce provádí nepřetržitě až do uvolnění tlačítka.

Akce, které lze ve hře ovládat, jsou následující:

- akcelerace lodě
- otáčení lodě doleva (u myši pohybem)
- otáčení lodě doprava (u myši pohybem)
- střelba ze zbraně
- přesun energie do štítů a naopak

Akcelerace a otáčení se po stisku tlačítka provádí nepřetržitě. Střelba a přesun energie se provede vždy jen jednou.

4.9.1 Ovládání pomocí klávesnice

Při ovládání pomocí klávesnice je každé akci přiřazena jedna klávesa. Při akceleraci a otáčení se při držení klávesy přičítá určitý přírůstek k aktuální hodnotě změny rychlosti či úhlu natočení (v závislosti na čase mezi snímky). Je vhodné umožnit v rámci parametrizace (kap. 4.11) změnu této hodnoty (citlivosti), aby si každý hráč mohl nastavit sobě vyhovující citlivost.

U klávesnice může nastat problém u stisku více kláves současně. Pokud hrají oba hráči na klávesnici, můžou být v jednu dobu najednou stisknuty až čtyři klávesy, a další čtyři mohou být ještě stlačovány přerušovaně. A toto znamená u mnoha klávesnic problém, jelikož většina registruje najednou pouze tři libovolné klávesy. Klávesnice je totiž uvnitř uspořádána do řádkových a sloupcových vodičů, které jsou různě sdílené, a klávesy jsou umístěny na jejich průnicích. Pokud stiskneme najednou dvě klávesy, každá z nich aktivuje jeden řádkový a jeden sloupcový vodič, a při stisku třetí klávesy může nastat jedna z těchto situací: buď klávesa aktivuje jiný řádkový nebo sloupcový vodič, než mají dvě již stisknuté, a bude rozpoznána, anebo aktivuje stejný řádkový vodič jako jedna ze stisknutých kláves a stejný sloupcový jako druhá, a nebude rozpoznána [1].

Řešením je vybírat klávesy tak, aby tyto kolize nenastávaly, nebo využít speciálních, dražších herních klávesnic, které toto řeší jinou, hardwarovou formou. Jako nekolizní klávesy se osvědčily kombinace znázorněné v tabulce 4.2.

Akce	**Hráč 1**	**Hráč 2**
akcelerace	W	šipka nahoru
otáčení doleva	A	šipka doleva
otáčení doprava	D	šipka doprava
střelba	levý shift	enter
přesun energie	Q	pravý shift

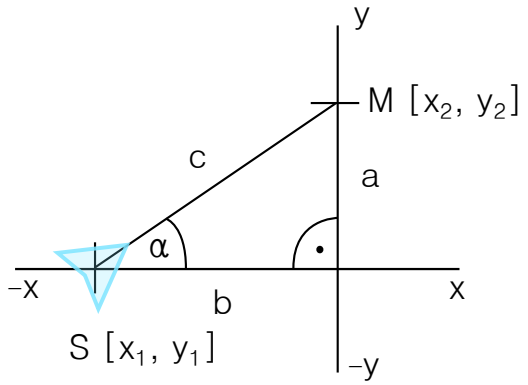
Tabulka 4.2: Přednastavené klávesy s menší pravděpodobností kolize.

Tyto klávesy jsou také přednastaveny jako implicitní (viz příloha A.3). Na různých klávesnicích ale kolize nelze vyloučit.

4.9.2 Ovládání pomocí myši

U myši není třeba řešit klávesy pro otáčení, jelikož to se děje přímo jejím pohybem. Tudíž lze zanedbat i citlivost, jelikož loď okamžitě reflektuje pozici myši. Ostatní akce je podobně jako u klávesnice třeba namapovat na jednotlivá tlačítka, popř. kolečko. Přednastavené mapování je: levé tlačítko – střelba, pravé tlačítko – akcelerace, prostřední tlačítko – přesun energie (viz příloha A.3).

Otáčení lodi na základě pozice myši spočívá v tom, že se mezi kurzorem myši a aktuální pozicí lodi spočítá úhel (obr. 4.6), který je pak lodi nastaven jako její aktuální rotace.



Obrázek 4.6: Úhel (α) mezi pozicí lodě (S) a pozicí kurzoru myši (M).

$$\text{tg}\ \alpha = \frac{a}{b} \tag{4.21}$$

$$\alpha = \text{tg}^{-1}(\frac{a}{b}) \tag{4.22}$$

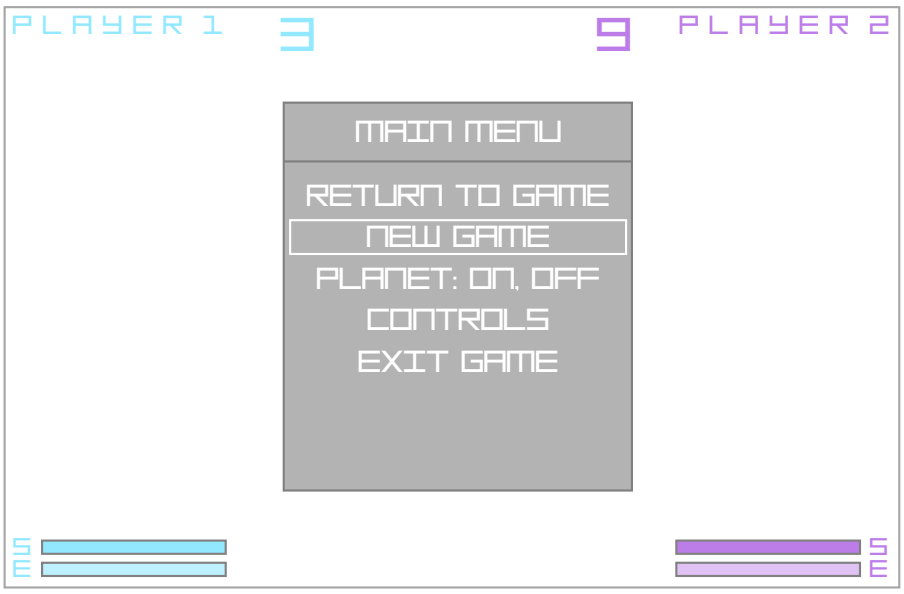
$$\alpha = \text{tg}^{-1}(\frac{y_2 - y_1}{x_2 - x_1}) \tag{4.23}$$

Výpočet úhlu je založen na zkoumání pravoúhlého trojúhelníku pomocí goniometrických funkcí. Použijeme funkci tangens, která je definována jako podíl délky protilehlé (a) a délky přilehlé (b) odvěsny trojúhelníku (vztah 4.21). Jelikož hodnoty stran a a b známe, úhel získáme použitím inverzní funkce arkus tangens (tg^{-1}, vztah 4.22). Nakonec dosadíme délky stran a a b (vztah 4.23). Protože je perioda funkce tangens $k\pi$, nepokryje celých 360 °, ale jen 180 °. Proto je nutné v druhém a třetím kvadrantu ($x_2 - x_1 < 0$) k vypočtenému úhlu ještě přičíst hodnotu 180 °.

4.10 Grafické uživatelské rozhraní

Grafické uživatelské rozhraní je potřeba pro infromaci hráčů o aktuálním stavu hry a k jejímu nastavování pomocí herní nabídky (menu). Na tvorbu komplexního uživatelské rozhraní existuje mnoho knihoven, například CEGUI[7], v našem případě si ale vystačíme s jednoduchým, ručně implementovaným rozhraním.

Rozhraní obsahuje výpis textů, ukazatele stavu a herní nabídku. Návrh rozhraní je vidět na obrázku 4.7. V dolních rozích jsou obsaženy ukazatele stavu. V horní části jsou texty identifikující jednotlivé hráče a aktuální hodnoty jejich skóre. Uprostřed je zobrazena herní nabídka, která se zobrazí po stisku klávesy `esc`.



Obrázek 4.7: Návrh grafického uživatelského rozhraní.

[7] Crazy Eddie's GUI System, viz http://www.cegui.org.uk.

4.10.1 Výpis textů

Výpis textů slouží k informaci o velikosti skóre každého hráče, různé informativní texty (viz obr. 4.7) a pro výpis jednotlivých položek herní nabídky. Text je také použit pro zobrazení stavu hry – informace, která se zobrazí v horním středu obrazovky a informuje o tom, který hráč právě bodoval.

OpenGL neposkytuje přímou podporu pro vytváření písem. Je třeba této podpory dosáhnout jinými metodami, které podporuje – kreslení bitmap, mapování textur s množinou znaků, vytváření čárových (*stroke*) písem či použití 3D geometrie pro každý jednotlivý znak[8].

V aplikaci je použita druhá zmíněná metoda – tzv. písmo z textury (*sprite font*). Tato metoda je jedna z nejvíce používaných, protože je přenositelná a rychlá. Ostatní metody většinou používají pro vytváření písem buď platformě závislé funkce operačního systému, anebo externí knihovny [3]. Případně lze písmo vytvořit úplně vlastními silami, tzn. ručním nadefinováním jednotlivých tvarů, ale toto je zbytečně pracná metoda.

Princip použité metody spočívá v tom, že je nejprve v grafickém editoru vytvořen rastrový obrázek s množinou všech znaků, které budou v aplikaci použity, a to v pořadí, v jakém jsou umístěny v ASCII tabulce (pro jednodušší následnou manipulaci při výpisu textů). Všechny znaky musí mít stejnou velikost a rozestupy (viz obrázek 4.8 použitý v této aplikaci). Následně je obrázek načten do programu jako textura. Z této textury se pak na základě pevných rozestupů namapují jednotlivé znaky na množinu geometrických primitiv, které jej budou reprezentovat (nejčastěji čtverce). Následně lze primitiva vykreslovat a tím tvořit texty. Aby se dosáhlo odstranění pozadí z každého znaku (průhlednosti), je využito míchání barev s alfa kanálem.

Obrázek 4.8: Sprite pro vytvoření písma.

Pro jednodušší a rychlejší vykreslování celých textů je v OpenGL využito *display listu*. Každé primitivum s namapovaným znakem se, v pořadí odpovídající ASCII tabulce, uloží do samostatného seznamu. Pro výpis řetězce pak jednoduše stačí předat jeho obsah do funkce `glCallLists`, která pro každý znak zavolá jeho příslušný seznam [3].

4.10.2 Ukazatele stavů

Ukazatele stavů (*progress bar*) slouží pro informaci hráčů o aktuální hodnotě energie a štítů jejich lodí. Jsou vykreslovány na základě dvou obdélníkových primitiv – vnějšího ohraničení (pouze z čar) a vnitřní výplně. Vnitřní výplň svou mírou vyplnění ohraničení (po ose x) určuje poměr aktuálně zobrazované veličiny vzhledem k jejímu rozsahu.

Veličinu je tedy nutno přepočíst na poměr výplně ukazatele. Ohraničení ukazatele je na herní ploše určeno svou délkou (O_{len}) – rozdílem koncového a počátečního bodu v ose x.

[8] Podrobnosti o jednotlivých metodách lze nalézt na http://www.opengl.org/resources/faq/technical/fonts.htm.

Veličina je určena svou minimální (V_{min}) a maximální (V_{max}) možnou hodnotou a hodnotou aktuální (V_{cur}), kterou chceme v ukazateli zobrazit. Potřebujeme tedy zjistit délku výplně ohraničení ukazatele(U_{len}) na základě této hodnoty. Přepočet je je znázorněn vztahem 4.24.

$$U_{len} = (O_{len}/(V_{max} - V_{min})) \cdot (V_{cur} - V_{min}) \tag{4.24}$$

Výsledná délka výplně ukazatele tedy odpovídá poměru zobrazované hodnoty k jejímu rozsahu.

Barva ohraničení ukazatele je měnena na červenou v případě kritických hodnot – pokud je hodnota štítů nižší než desetina jejich maximální hodnoty, anebo když hodnota energie nestačí na výstřel z aktuální zbraně (v případě výstřelu z bonusové zbraně při nedostatku energie je pak tato zbraň lodi odebrána).

Nad ukazateli je v při aktivace bonusové zbraně zobrazován tvar bonusu, pomocí kterého byla zbraň aktivována. Tento tvar rotuje stejně jako bonusy na hrací ploše a vedle něj je vypsán text s názvem bonusové zbraně. Barva obou elementů je změněna z původní barvy bonusu do odstínu barvy hrače, kterému je bonusová zbraň přiřazena.

4.10.3 Herní nabídka

Herní nabídka je zobrazitelná pomocí stisku klávesy `esc`. Když je zobrazena, hra se pozastaví (herní objekty se přestanou aktualizovat) a veškeré vstupy z klávesnice či myši jsou presměrovávány pouze jí. Nabídka reaguje na pohyb myši a kurzorové klávesy (označování položek), potvrzení právě označené položky se pak provede buď klávesou `enter`, anebo stiskem levého tlačítka myši nad označenou položkou.

Vzhled herní nabídky je zachycen na obrázku 4.7. Jednotlivé položky pak mají následující význam:

Return to game slouží pro skrytí nabídky a návrat do hry.

New game spouští novou hru. To znamená, že se vynuluje hodnota skóre a lodě jsou umístěny na jejich základní pozice.

Planet: on, off určuje nastavení planety. Část za dvojtečkou ukazuje její aktuální nastavení. První parametr určuje, zda je planeta vůbec přítomna. Druhý parametr určuje zapnutí gravitace. Pokud je gravitace vypnuta, planeta se neanimuje (neotáčí se). Kombinace s vypnutou planetou a zaplou gravitací není možná (*off, on*). Při potvrzení položky se nastavení přepne do další možné kombinace (celkem tři).

Controls zobrazí podnabídku pro nastavení ovládání. Ta obsahuje volby pro nastavení ovladače (klávesnice/myš) každému hráči. Myš nelze nastavit současně oběma hráčům. Dále jsou obsaženy podnabídky pro nastavení jednotlivých kláves a tlačítek. Podnabídky jsou celkem čtyři, pro každého hráče dvě (nastavení kláves a tlačítek zvlášť).

Podnabídka s nastavením kláves/tlačítek vypíše jako své položky jednotlivé akce s názvy právě přiřazených kláves. Pokud uživatel nějakou akci vybere a potvrdí, zobrazí se informace o tom, že hra čeká na novou klávesu pro její přiřazení. Po stisku jakékoli klávesy se tato klávesa vybrané akci přiřadí. Zrušení zadávání lze zvolit stiskem klávesy `esc` – tu pochopitelně akcím přiřazovat nelze. Obě nabídky obsahují volbu pro nastavení implicitních hodnot (*Reset to defaults*).

Exit game ukončí hru (vypne celou aplikaci).

Nabídka je vykreslována pouze jako obdélníkové primitivum s využitím výše zmíněného výpisu textů. Pohyb v nabídce a jejich jednotlivých úrovních, včetně změn ovládání, je založen na konečném automatu. Návrat z podnabídek o úroveň výše se provede buď vždy obsaženou poslední volbou *Back*, anebo stiskem klávesy `esc`. Na nejvyšší úrovni stisk této klávesy způsobí skrytí nabídky.

4.11 Parametrizace

Aby hra byla co nejvíce škálovatelná, je možno upravovat vybrané parametry pomocí konfiguračních souborů. Konfigurační soubor je textového formátu jednoduchého stylu `klíč=hodnota`. Souborů je vytvořeno více (celkem čtyři), aby každý z nich pokrýval samostatnou oblast nastavení. V případě smazání souborů či nenalezení konfigurované hodnoty se načtou hodnoty implicitní a soubor je po ukončení aplikace znovuvytvořen.

4.11.1 Herní vlastnosti

Konfigurace herních vlastnosti se dělí do čtyř logických částí. Každá z částí obsahuje více konkrétních nastavení. Části jsou následující:

- vlastnosti lodí (stejné pro obě)
- nastavení planety
- nastavení systému bonusů
- vlastnosti zbraní (pro každou zbraň samostatně)

Všechny konkrétní hodnoty, které lze upravovat, jsou uvedeny v příloze A.4.

4.11.2 Barvy

V tomto souboru je uložena většina barevného nastavení ve formátu `R,G,B,A`, kde písmena značí jednotlivé barevné složky (červená, zelená, modrá, alfa kanál) v hodnotách `0-255`. Lze tak měnit barvy většiny objektů – lodí, střel, částicových efektů, bonusů, planety, části grafického uživatelského rozhraní a objektů na pozadí.

4.11.3 Zobrazení a ovládání

Poslední dva konfigurační soubory pokrývají systémovější nastavení – vlastnosti herního okna (příloha A.2) a ovládání (příloha A.3). U herního okna lze nastavit rozlišení a zvolit, zda má být aplikace zobrazena přes celou obrazovku. Také lze vypínat a zapínat vertikální synchronizaci obrazu. Nastavení ovládání je zde jen pro úplnost, jelikož lze měnit v rámci herní nabídky (kap. 4.10.3) a tudíž je editace přímo tohoto souboru zbytečná. Soubor obsahuje aktuálně přiřazené kódy kláves a tlačítek ke všem jednotlivým akcím každého hráče a informaci o tom, jaký typ ovladače (klávesnice/myš) každý z hráčů aktuálně používá.

Kapitola 5

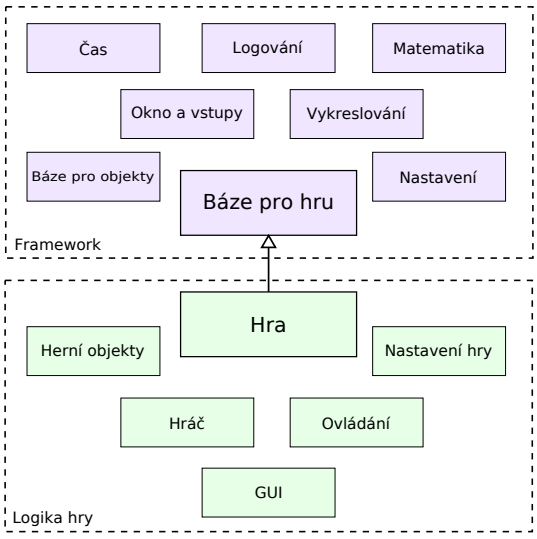
Implementace

Při implementaci aplikace byl od začátku kladen důraz na objektový návrh se snahou využití některých návrhových vzorů. S tímto přístupem by měla být jednoduchá udržovatelnost, znovupoužitelnost a rozšiřitelnost kódu.

5.1 Architektura a objektový návrh

Hra byla vytvořena jako klasická okenní aplikace využívající knihovnu SDL pro vytvoření okna a správy jeho údálostí, včetně správy vstupů z klávesnice a myši. Pro vykreslování akcelerované grafiky pak byla použita knihovna OpenGL.

Základní architekturu aplikace zachycuje obrázek 5.1. Aplikace byla rozdělena do dvou hlavních částí – herního *frameworku*[1], který poskytuje základní obecnou funkčnost, jako správu okna a vstupů, vykreslování, přístup k časovačům apod., a konkrétní implementaci hry, která využíva služeb toho frameworku. Samotná logika hry je už přímo ve zděděné třídě. Tento přístup byl inspirován Microsoft XNA Frameworkem [12].



Obrázek 5.1: Základní architektura aplikace.

[1]Framework je softwarová struktura, která slouží jako podpora při programování.

Programování probíhalo oproti rozhraním a abstraktním třídám, aby byly jednotlivé implementační detaily jednoduše zaměnitelné. To znamená, že by například měla být jednoduše nahraditelná vykreslovací část (použití jiné knihovny), popř. jiný správce oken, bez vlivu na logiku hry.

5.2 Herní framework

Aby se odstínily některé obecné a platformě závislé funkce od konkrétní implementace hry, byla vytvořena tato část. Obsahuje osm logických celků, každý z nich je implementován jednou anebo více třídami. Všechny třídy patřící do této části jsou obsaženy v prostoru jmen (*namespace*) `GameFramework`. Každá ze tříd je zpravidla obsažena v samostatném modulu (*.cpp a *.h) stejného názvu, jednodušší jsou obsaženy pouze v hlavičkovém souboru (*.h).

5.2.1 Správa okna a vstupů

Tento celek je základní stavební kámen aplikace, umožňuje vytvořit okno a reagovat na jeho události. Obsahuje jedno rozhraní a tři abstraktní třídy, které jsou pak implementovány konkrétní knihovnou, v našem případě SDL.

RenderWindow je abstraktní třída spravující vykreslovací okno. Obsahuje metodu `Create`, která vytváří okno se zadanými parametry (velikost, celá obrazovka, apod.). Oknu je možno měnit velikost. Umožňuje metodou `SetEventCallback` registrovat třídu pro příjem událostí přicházejících oknu (tato třída pak musí implementovat rozhraní `IWindowEventListener`, viz dále). Třída také pro zjednodušení zaštiťuje správu vstupů (`Keyboard` a `Mouse`, viz níže), jelikož vstupy jsou také události okna. Pro příjem a zpracování událostí je třeba pravidelně volat metodu `ProcessEvent`.

Konkrétní implementace pomocí knihovny SDL je ve třídě `SDLWindow`. Zpracování událostí zajišťuje funkce `SDL_PollEvent`, která vyjme z fronty aktuální událost. Na základě jejího typu se poté zavolá konkrétní metoda z rozhraní `IWindowEventListener` třídy registrované k odběru událostí.

IWindowEventListener je rozhraní (obsahuje pouze čistě virtuální metody) deklarující metody pro příjem událostí z okna. Metody jsou následující:

- změna velikosti okna
- stlačena klávesa
- uvolněna klávesa
- stlačeno tlačíko myši
- uvolněno tlačítko myši
- pohyb myší

Třída implementující toto rozhraní se u třídy `RenderWindow` může zaregistrovat k odběru událostí. Při příchodu události se poté zavolá konkrétní metoda v implementující třídě.

Keyboard a **Mouse** jsou abstraktní třídy pro jednotnou práci s klávesnicí a myší. Pracují s obecnými výčty kódů kláves/tlačítek (`KeyCode` a `BtnCode`), protože v různých implementacích se kódy kláves a tlačítek liší. Třídy si pomocí pole udržují informaci

o tom, které všechny klávesy/tlačítka jsou právě stisknuta (díky spolupráci s metodou `ProcessEvent`). Pomocí této vlastnosti lze pak ve hře detekovat více stisklých kláves/tlačítek současně a také provádět akce na základě držení kláves/tlačítek, ne jenom jejich stisku. Umožňují také, na základě předaného obecného kódu, vrátit jejich textový název.

Konkrétní implementace pomocí knihovny SDL jsou ve třídách `SDLKeyboard` a `SDLMouse`. Každá ze tříd obsahuje asociativní pole, která mapují obecné kódy kláves/tlačítek na konkrétní kódy použité knihovnou. Textový název klávesy se zjišťuje pomocí funkce `SDL_GetKeyName`. Názvy tlačítek jsou přiřazeny ručně, protože knihovna SDL obdobnou funkci jako u kláves neobsahuje.

5.2.2 Vykreslování

Druhý z hlavních celků zajišťuje (a také zjednodušuje) veškeré vykreslování grafiky na obrazovku. Obsahuje abstraktní třídy, které jsou poté implementovány konkrétní grafickou knihovnou, v našem případě OpenGL. Také obsahuje další, pomocné třídy.

Color je třída pro jednotnou reprezentaci barvy. Používá se všude tam, kde se barvou pracuje. Je vnitřně reprezentována jako čtveřice hodnot typu `float`, ale umožňuje i přepočet z/do celočíselné reprezentace o rozsahu 0–255. Také obsahuje metodu na lineární interpolaci mezi dvěmi barvami.

Shape reprezentuje tvar, který je možno vykreslit. Vnitřně je reprezentován jako pole vrcholů. Toto pole lze nastavovat přednastavenými metodami. Jdou tak například přidat body tvořící čáru, obdélník, ale i trojrozměrný hranol či jehlan. Lze také zvolit styl vykreslování – jednotlivé čáry, polygon, anebo jen body. Při vykreslování je pak tvar v OpenGL vykreslen pomocí pole vrcholů (*vertex array*) funkcí `glDrawArrays`.

Image je pomocná abstraktní třída pro reprezentaci obrázku. Umožňuje jeho načtení a získání bitmapy ve formě pole. Konkrétní formáty obrázků jsou implementovány v odvozených třídách. Instance konkrétních typů obrázků vytváří jednoduchá tovární metoda (*simple factory method*, viz [14]) `CreateImage` obsažená ve třídě `ImageFactory`. Metoda vrátí novou instanci konkrétního typu obrázku na základě jeho přípony v předané cestě. Lze tak jednoduše přidávat podporu nových formátů obrázků bez jakéhokoli zásahu do ostatního kódu. Konkrétně je vytvořena podpora obrázků typu BMP ve třídě `BMPImage`. Používá se pro náčítání textur.

Texture je abstraktní třída představující texturu načtenou v konkrétní grafické knihovně. Umožňuje její načítání pomocí třídy `Image` a následné svázání s vykreslovaným objektem. Konkrétní OpenGL implementace je ve třídě `OpenGLTexture`.

SpriteFont je abstraktní třída reprezentující písmo z textury. Metodou `Create` se na základě cesty k souboru s texturou vytvoří vnitřní reprezentace, pomocí které lze pak vykreslovat znakové řetězce. Metodě je nezbytné předat parametry určující počet a velikost znaků obsažených na textuře. Třída také obsahuje metody pro zjištění délky a výšky vypisovaného řetězce v pixelech, což je potřeba například pro přesné umisťování textů na určitou pozici.

Konkrétní OpenGL implementace je obsažena ve třídě `OpenGLSpriteFont`. Na základě cesty se nejprve vytvoří textura. Poté je vytvořen *display list* pomocí třídy

`OpenGLDisplayList` a do něj jsou vygenerovány obdélníky s namapovanými výřezy textury představující jednotlivé znaky. Písmo je následně vykreslováno pouze na základě volání tohoto display listu.

Renderer je základní abstraktní třída pro veškeré operace spojené s vykreslováním. Obsahuje metody pro inicializaci vykreslovací knihovny, její nastavování a hlavně vykreslování. Umí vykreslit čáru, kouli, zadaný tvar či řetězec. Také obsahuje tovární metody pro vytváření textur a písma z textury. Umožňuje také nastavovat aktuální projekční a model-pohledové matice.

Konkrétní implementace knihovnou OpenGL je ve třídě `OpenGLRenderer`. Koule je vykreslována pomocnou knihovnou GLU (*OpenGL Utility Library*).

5.2.3 Práce s časem

Pro jednodušší práci s časem bylo vytvořeno několik tříd. Práce s časem je důležitá hlavně pro počítání doby mezi dvěmi snímky a pro děje, které jsou závislé na časovém intervalu.

TimeSpan je třída reprezentující časový údaj. Údaj je možno zadávat ve formátu sekund či milisekund. Vnitřně je reprezentován jako počet milisekund.

GameTime je třída představující herní čas. Obsahuje celkový čas od spuštění aplikace, aktuální čas a rozdílový čas mezi aktuálním a předchozím snímkem. Instanci této třídy je proto nutné aktualizovat v každém snímku metodou `Update`. Aktuální čas se zjišťuje ze třídy `SystemTime`, která vrací celkový čas od zapnutí počítače za použití funkce `SDL_GetTicks`.

FpsCounter počítá aktuální počet snímků za sekundu a udžuje informace o nejlepší, nejhorší a průměrné hodnotě. Tato třída je zde zmíněna jen pro úplnost, protože s časem přímo nesouvisí, využívá jen informaci o jeho aktuální hodnotě a její použití se hodí jen pro informativní charakter o tom, jak rychle hra běží.

Instance třídy musí být aktualizována v každém snímku metodou `Update`. Uvnitř metody je inkrementována hodnota udávající aktuální počet snímků. Po uběhnutí jedné sekundy je výsledná hodnota spočtena jako poměr aktuálního počtu snímků a uplynutého času (v tomto případě sekundy – lze zanedbat), čítač je vynulován a proces se opakuje. Průměrná hodnota se pro jednoduchost počítá jen z aktuální a předchozí hodnoty.

5.2.4 Podpora matematiky

Tato část je obsažena pro pohodlnější práci s často používanými matematickými operacemi, které jsou pří vývoji her většinou nezbytné.

Vector2 a **Vector3** jsou šablonové struktury reprezentující dvourozměrný, respektive třírozměrný vektor. Struktury obsahují základní vektorové operace, jako je délka a normalizace. Díky přetíženým operátorům (včetně skalárního součinu) je lze intuitivně používat. Lze tak například jednoduše zjistit kolizi mezi dvěmi kružnicemi se středy určeným vektory `vec` a poloměry `radius`:

```
(vec1 - vec2).Length() <= radius1 + radius2
```

Pro jednodušší používání byly nadefinovány konkrétní šablonové typy – `Vector2f` pro reprezentaci typem `float` a `Vector2i` pro reprezentaci typem `int`. Také byly přetíženy operátory přetypování, takže lze mezi těmito typy i jednoduše přetypovávat.

Matrix reprezentuje matici hodnot typu `float` o rozměru 4×4. Bylo by možno pro maticové operace využít vestavěné funkcionality OpenGL[2], ale pro jednodušší manipulaci byla vytvořena třída vlastní. Třída pracuje s reprezentací matice ve stejném formátu jako OpenGL. Umožňuje matice násobit, nastavovat transformace (posunutí, rotace, změna měřítka) a také projekce. Funkcionalita těchto metod byla vytvořena podle implementace v OpenGL [13]. Nastavení matice jako aktuální se provede ve vykreslovací části (v implementaci pomocí OpenGL funkcí `glLoadMatrixf`).

MathHelper je knihovní třída (*library class*, viz [14]) obsahující obecné matematické funkce sloužící jako podpora některým často používaným výpočtům, například převod stupně na radiány a naopak. Také obsahuje funkce pro zjišťování průníků mezi objekty – dva kruhy a kruh s úsečkou.

Random zajišťuje generování pseudonáhodných čisel s rovnoměrným rozložením. Základem pro generování je funkce `rand` ze standardní knihovny. Třída obsahuje také přepočty do zadaných rozsahů, takže lze například jednoduše generovat náhodne číslo se zadaným rozsahem `min` a `max`, a to i s výběrem mezi verzí celočíselnou a verzí s plovoucí řádovou čárkou.

5.2.5 Nastavení

Tato část slouží pro správu nastavení. Umožňuje načítání a ukládání hodnot z/do konfiguračních souborů.

KeyValueSerializer a **KeyValueDeserializer** jsou třídy pro ukládání/načítání hodnot ve formátu `klíč=hodnota` do/z textových konfiguračních souborů. Třídy využívají specializaci šablon pro jednotné ukládání hodnot různých objektů.

Při ukládání se hodnoty nejprve uloží do vnitřního pole (příkaz `Save`) a až po zavolání metody `WriteToFile` se uloží do souboru.

V případě načítaní se všechny hodnoty nejprve najednou načtou ze souboru do asociativního pole (metoda `ReadFromFile`). Poté je možné se na tyto hodnoty dotazovat metodou `Load`, která ve svém druhém parametru vrátí hodnotu odpovídající klíči. Pokud nebude hodnota nalezena, zaloguje se informativní zpráva a metoda vrátí hodnotu předanou jako třetí parametr (považovanou za implicitní).

Settings je abstraktní třída představující nastavení v jednom konfiguračním souboru. Pro nahrávání a ukládání konfigurace bylo využito návrhového vzoru *template method* (šablonová metoda, viz [14]). Odvozená třída musí implementovat metody pro načtení implicitních hodnot a pro načtení/uložení konkrétního nastavení. Každá odvozená třída si vytvoří vlastní strukturu s hodnotami, které chce ukládat. Po zavolání metody `Load`/`Save` (obsažené v této třídě) se na základě šablonové metody veškeré hodnoty z potomka načtou/uloží do zvoleného konfiguračního souboru. Pokud při načítání soubor neexistuje, načtou se implicitní hodnoty a při následném uložení se soubor vytvoří.

[2] Od OpenGL verze 3.1 byla podpora maticových operací odstraněna, viz http://www.opengl.org/wiki/Viewing_and_Transformations. Aplikace si jí musí zajistit vlastními metodami.

FrameworkSettings dědí ze třídy `Settings` a představuje konkrétní nastavení frameworku uložené v souboru `framework.cfg`. Obsahuje zvoleného správce okna a grafickou knihovnu a jejich vlastnosti – velikost okna, zda má být přes celou obrazovku a volbu vertikální synchronizace obrazu.

5.2.6 Logování

Tato část obsahuje pouze jedinou třídu – `Log`, která slouží k zaznamenávání událostí do textového souboru – logu. Logování je důležité pro různé informace o běhu programu, které mohou být užitečné například při zjišťování chybných stavů.

Třída je vytvořena pomocí návrhového vzoru *singleton* (jedináček), aby byl možný k její instanci globální přistup ze všech ostatních objektů [14]. Obsahuje dvě hlavní metody – `Msg` a `Stream`. `Msg` si bere za parametr řetězec, který následně zaloguje. `Stream` je metoda vracející vnitřní objekt `LogStream`, pomocí něhož lze zprávy do logu zapisovat stejně jako do výstupního proudu – lze jednoduše logovat číselné hodnoty bez nutnosti jejich explicitniho převodu na řetězec. `LogStream` obsahuje pomocnou vnitřní proměnnou typu `std::stringstream` (ze standardní knihovny), do které proud zapisuje. Při detekci konce proudu se vytvořená zpráva zaloguje. Všechny logované zprávy se zapisují s aktuálním časovým údajem ve formátu HH:MM:SS do textového souboru `game.log`.

Protože při zápisu logované hodnoty se hodnota nezapíše do souboru hned, ale nejprve do vyrovnávací paměti, je třeba po každém zápisu volat funkci `flush`. Tato funkce vynutí zapsání obsahu paměti do souboru. Je to důležité pro případy, kdy program spadne a informace o chybě se ještě nestihly na disk zapsat a byly ztraceny.

5.2.7 Báze pro herní objekty

Tento celek obsahuje základní rozhraní pro odvozování herních objektů. Herním objektem se rozumí objekt, který umí aktualizovat svůj stav a vykreslit se. Tyto rozhraní byly vytvořeny proto, aby se s těmito objekty dalo jednotně pracovat.

IUpdatableObject představuje objekt, který pravidelně aktualizuje svůj stav. Obsahuje jedinou čistě virtuální metodu `Update`, která si jako parametr bere konstantní odkaz na instanci objektu `GameTime` (herní čas). Ten je potřeba znát pro stabilní rychlost pohybu (viz kap. 4.4.3) a také pro operace závislé na časovém intervalu. Tato metoda je také zpravidla místo, které určuje veškerou logiku objektu (jeho chování ve hře).

IDrawableObject představuje objekt, který se umí vykreslit. Obsahuje jedinou čistě virtuální metodu `Draw`, která si jako parametr bere odkaz na instanci objektu `Renderer`, pomocí jehož metod může vykreslení provést. V této metodě by se měly provádět pouze akce související s vykreslováním aktuálního stavu objektu, který může být měněn metodou `Update`. Tím je odděleno vykreslování od logiky objektu – lze tak například jednoduše zapauzovat hru (přestat volat funkci `Update`) a pozastavené objekty dále vykreslovat.

IBasicGameObject pouze spojuje dvě předchozí metody do jednoho rozhraní (je z předchozích rozhraní odvozen). Obě metody byly původně na dvě samostatná rozhraní rozděleny proto, protože můžou existovat objekty, které provádějí jen jednu akci (jen aktualizaci nebo jen vykreslování).

5.2.8 Báze pro hru

Báze pro hru spojuje většinu předešlých celků a obsahuje základní životní cyklus aplikace. Konkrétní implementaci herní logiky pak nechává na zděděné třídě (návrhový vzor template method).

Implementace je v jediné třídě `Game`, ze které pak konkrétní implementace hry dědí. Třída obsahuje instance objektů herního okna, vykreslovací části, herního času, čítače snímků za sekundu a nastavení. Také implementuje rozhraní `IWindowEventListener`, aby bylo možné reagovat na zprávy přicházející oknu. Vstupní bod tvoří jediná veřejná metoda `Run`, ve které se provede kompletní inicializace a poté spuštění herní smyčky.

V metodě se nejprve vytvoří instance třídy `Log`, aby bylo hned od začátku možné logování událostí. Následně se zavolá metoda `BeforeCreate`, ve které se načte nastavení frameworku (třída `FrameworkSettings`). Poté se provede inicializační metoda `Initialize`, ve které se na základě nastavení vytvoří herní okno, vykreslovací část a inicializuje se herní čas. Třída se také přihlasí oknu k odběru jeho událostí (`SetEventCallback`). Chybové stavy se detekují na základě odchytu vyjímek, které se zalogují a aplikace je předběžně ukončena. Dále už přichází hlavní smyčka aplikace, která vždy nejprve zpracuje aktuální události došlé oknu (metoda `ProcessEvent`) a poté zavolá metodu `Tick`, která představuje jednu aktualizaci aplikace. V případě, že oknu dojde událost o jeho zavření (`ProcessEvent` vrátí `false`), anebo si vyžádáme zavření sami (metoda `Exit`), provede se deinicializační metoda `Deinitialize`, ve které se uloží aktuální nastavení frameworku a odstraní se vykreslovací část a okno aplikace. Nakonec je odstraněn log a aplikace je ukončena.

V aktualizační metodě `Tick` se nejprve aktualizuje herní čas, na jeho základě se aktualizuje čítač snímků za sekundu (`FpsCounter`) a následně se zavolají funkce `Update` (s parametrem aktuálního herního času a čítače snímků) a `Draw`. V těchto funkcích se pak ve zděděné třídě provádí aktualizace a vykreslování herních objektů.

Všechny metody lze v potomkovi překrýt (*override*) vlastní implementací. Pokud chceme jejich implementaci pouze rozšířit, nesmíme v nich zapomenout implementaci rodičovskou zavolat.

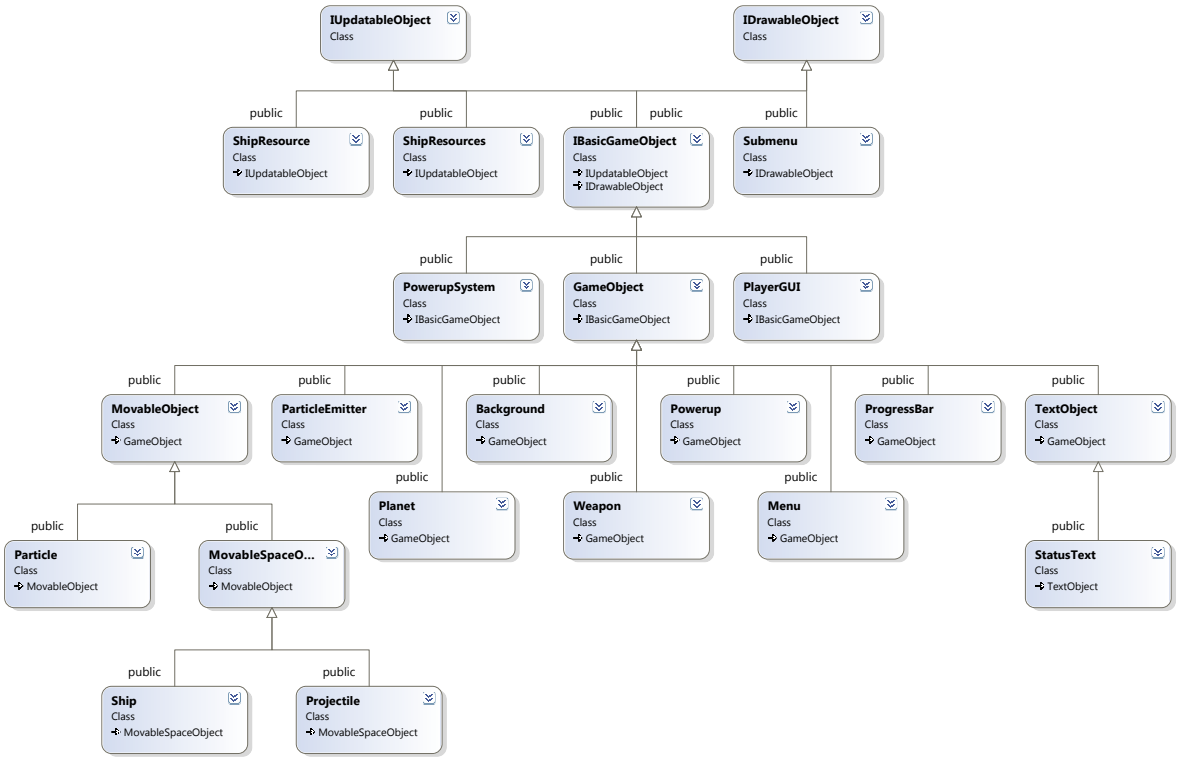
5.3 Logika hry

Tato část obsahuje konkrétní logiku hry využívající služeb herního frameworku. Využití frameworku je dosaženo na základě odvození (dědění) z jeho tříd. Všechny třídy patřící do této části jsou obsaženy v prostoru jmen `RetroSpace`. Stejně jako u herního frameworku je zpravidla každá ze tříd umístěna v samostatném modulu, jednodušší pak pouze v hlavičkovém souboru.

5.3.1 Základní herní objekty

Základní herní objekty slouží jen jako bázové třídy ostatním herním objektům. Obsahují jejich společnou funkčnost a tím s nimi značně zjednodušují práci. Vzájemná dědičnost všech herních objektů je pak znázorněna na obrázku 5.2.

GameObject je základní třída pro herní objekt. Implementuje rozhraní `IBasicGameObject`, takže objekt lze aktualizovat a vykreslovat. Obsahuje atributy určující jeho pozici, rotaci (ve stupních) a měřítko. Také obsahuje matici (třída `Matrix`) pro maticovou reprezentaci jeho transformace (aby bylo možno transformace jednoduše skládat). Matice se vytvoří podle výše zmíněných atributů



Obrázek 5.2: Dědičnost herních objektů.

metodou `UpdateMatrix` a následně ji lze nastavít jako aktuální do vykreslovací části metodou `ApplyMatrix`. Tomuto objektu lze také testovat kolize s jiným objektem na základě průniku dvou kruhů (metoda `Collide`). Poloměr je určen atributem `m_collideRadius` a jeho implicitní hodnota je nastavena na velikost deseti bodů.

MovableObject dědí z `GameObject` a rozširuje jeho funkčnost o možnost pohybu. Přidává atributy udávající rychlost (vektor `m_velocity`) a její změnu (vektor `m_acceleration`). V metodě Update se poté na základě těchto hodnot a rozdílu času mezi dvěma snímky aktualizuje pozice objektu – objekt se pohybuje (viz kap. 4.4.2). Třída také obsahuje metody pro nastavení zadané rychlosti do směru určeného úhlem včetně opačné operace – rotace objektu ve směru jeho pohybu.

MovableSpaceObject rozšiřuje funkcionalitu `MovableObject` o dva vlivy – gravitaci a udržování objektu na hrací ploše (v případě dosažení kraje plochy se přesune na kraj opačný). Objekt tedy potřebuje znát dvě informace: velikost hrací plochy a vlastnosti planety (polohu, sílu a aktivitu gravitace apod.). Tyto hodnoty se zjišťují na základě předaného odkazu na objekt planety (třída `Planet`).

5.3.2 Pozadí a planeta

Pozadí herní plochy a planeta jsou dva nejjednodušší herní objekty. Oba dědí z třídy `GameObject` a musí reagovat na změnu velikosti hrací plochy.

Background je třída představující pozadí. Zajišťuje vykreslování mřížky a bodů znázorňujících hvězdy. Mřížka i hvězdy se vykreslují za pomoci třídy `Shape`. Mřížka je složena

z úseček odpovídajících počtem a velikostí rozměrům hrací plochy. Pro generování hvězd je využita metoda `AddRandom` třídy `Shape`, která do tvaru přidá zadaný počet bodů (závislých na aktuální velikosti okna, aby jejich hustota byla vždy stejná) na náhodných pozicích. Při změně velikosti hrací plochy se hodnoty přegenerují, aby reflektovaly její aktuální velikost.

Planet reprezentuje planetu. Zajišťuje její vykreslení včetně animace rotace. Obsahuje atributy o síle a aktivitě její gravitace a také o tom, zda je planeta zapnutá. Pokud gravitace není aktivní, planeta se neanimuje. Pokud je vypnutá, není zobrazena a detekce kolizí s ní je vypnuta.

5.3.3 Částicový systém

Implementovaný částicový systém obsahuje celkem šest tříd, z toho čtyři slouží pro konfiguraci efektů.

Particle je třída reprezentující jednu částici, která je vykreslována jako jednoduchá čára. Dědí od třídy `MovableObject`, takže obsahuje možnost pohybu, které také využívá. V konstruktoru jsou jí nastaveny parametry probrané v kapitole 4.8.1 a v aktualizační funkci se podle nich chová, tzn. na základě své životnosti interpoluje barvu z počáteční do koncové a, pokud je nastaveno, zkracuje velikost čáry, kterou je reprezentována. Koncový čas životnosti `m_endTime` se nastaví v prvním průchodu aktualizační funkce, a to jako aktuální čas plus čas doby života (`m_lifetime`). Poté se aktuální čas porovnává s tímto koncovým časem, a když je větší, životnost částice končí (`m_alive=false`).

ParticleEffect je abstraktní třída pro konfiguraci vlastností skupiny částic. Odvozená třída pak představuje konkrétní konfiguraci efektu. Třída obsahuje metodu pro každý parametr částice. Každá z metod může svoji návratovou hodnotu generovat buď jako konstantní, nebo na základě generátoru pseudonáhodných čísel (třída `Random`), a nebo na základě pořadového čísla částice, nastavovaného metodou `SetParticleNumber`.

Pro konkrétní efekty byly tedy odvozeny dvě třídy, `ExplodeParticleEffect` pro efekt výbuchu a `TrailParticleEffect` pro efekt pohonu lodí a raket. Oba efekty se liší především v metodě určující rychlost (`Velocity`), a to hlavně její směr. Efekt pohonu navíc generování provádí ze dvou výchozích pozic (efekt dvou trysek). Toho je dosaženo pomocí pořadového čísla částice, kdy každou sudou generuje s určitým posunem po ose y (metoda `Position`).

Pro přehlednější vytváření konkrétních efektů byla vytvořena třída `ParticleEffectFactory` s jednoduchou tovární metodou `CreateParticleEffect`, která vrátí konkrétní implementaci efektu na základě předaného výčtu typu `Effect`.

ParticleEmitter slouží jako generátor částic konkrétního efektu a zároveň také jako jejich správce. To znamená, že se kromě vytváření stará i o jejich celý životní cyklus. Metodou `Emit` se na základě parametrem předaného efektu (`ParticleEffect`) vygeneruje zadaný počet částic. V cyklu jsou dynamicky vytvářeny nové instance objektu typu `Particle` s parametry efektu. Ukazatel na každou vygenerovanou částici je uložen do vnitřního seznamu (`std::list`), aby mohly být následně spravovány.

V aktualizační funkci se prochází pole vygenerovaných částic a volá se na ně jejich aktualizační funkce. Vždy je kontrolován parametr `m_alive` každé částice metodou

`IsAlive`, a pokud je nepravdivý, částice je dealokována a z pole odstraněna. Ve vykreslovací funkci se pole částic pouze prochází a volá se na ně metoda `Draw`. Po skončení života poslední částice v poli je nastaven vnitřní atribut `m_active` na hodnotu `false`, což udává, že generátor dokončil správu svých vygenerovaných částic.

5.3.4 Zbraně

Zbraně fungují na základě vystřelovaných projektilů zadaných vlastností. Princip chování je podobný jako u částicového systému.

Projectile je třída reprezentující jeden vystřelený projektil. Dědí od třídy `MovableSpaceObject`, takže na jeho pohyb působí navíc gravitace a je udržován na hrací ploše. Jeho chování je podobné chování jedné částice. V konstruktoru jsou mu také předávány atributy definující jeho vlastnosti, kterými se pak po dobu svého života řídí. Tyto vlastnosti jsou probrány v kapitole 4.5.2. Doba života je počítána stejně jako u částice. Tvar vykreslovaného projektilu je definován objektem typu `Shape`.

Třída překrývá metodu `Collide` bázové třídy `GameObject`, protože kolizi s projektilem je třeba v případě laseru zjišťovat na základě průniku úsečky a kružnice, ne jenom dvou kružnic. Projektil také obsahuje na základě parametrů dva generátory částic, jeden pro efekt výbuchu (kanón a raketa), druhý pro efekt trysek (raketa).

WeaponTypes slouží jako databáze atributů chování jednotlivých projektilů. Část jejich hodnot je načítána z konfiguračního souboru. Třída obsahuje pole atributů typu `Attributes`, což je struktura definující chování jednoho projektilu. Pole je indexováno výčtem `Type`, který určuje typ zbraně a tedy chování jejího projektilu.

Jednotlivé atributy v tomto poli jsou nastaveny v metodě `BuildTypes`. Metoda obsahuje také vytvoření tvarů jednotlivých projektilů, a také nastavení, jakou metodu použít pro detekci kolize. U kolize přímky a kružnice není nastavován poloměr, ale koncový bod úsečky relativně od středu projektilu. Dále jsou v metodě ještě alokovány a nastavovány jednotlivé částicové efekty (objekty typu `ParticleEffect`).

Veškeré nastavení těchto projektilů se pak předává do jiných tříd jen odkazem, kopírování by bylo zbytečné, protože nastavení projektilů je stejné pro oba hráče. Pro odstranění nutnosti předávání odkazů by třída mohla být implementována pomocí návrhového vzoru singleton.

Weapon je třída podobná třídě `ParticleEmitter`. Umožňuje generovat (vystřelovat) jednolivé projektily a poté spravovat jejich celý životní cyklus. Na základě parametrů projektilu také omezuje jejich použití – kontroluje například kadenci a nedovolí vystřelit dříve, než po uplynutí určeného časového intervalu. Správa vystřelených projektilů probíhá pomocí seznamu stejně jako u částic.

5.3.5 Systém bonusů

Systém bonusů je tvořen opět podobným způsobem jako výše – jedna třída pro konkrétní prvek, druhá pro správu jejich množiny.

Powerup představuje jeden bonus vygenerovaný na hrací ploše. Po vygenerování nijak nemění pozici, pouze se animuje způsobem popsaným v kapitole 4.7.1. V případě, že

je sebrán jednou z lodí, zavolá se metoda `OnPickUp` a přejde do stavu provádění efektu popsaného v kapitole 4.7.2. To znamená, že se k hodnotě udávající jeho měřítko začne přičítat konstanta násobená časovým rozdílem mezi snímky a zároveň se zobrazí na herní ploše text s názvem bonusu, který je posouván směrem vzhůru po ose y až do ztracena (do konce efektu – 1 000 ms). Po dokončení efektu je nastaveno `m_alive` na `false` a bonus je odstraněn.

PowerupSystem zajišťuje generování bonusů a správu jejich životního cyklu. Jednotlivé typy bonusů jsou vytvořeny v metodě `BuildPowerups`. Generování probíhá v aktualizační funkci `Update` po uplynutí časového intervalu udávajícího rozestup mezi generováním, nastavovaného v metodě `GenerateNext` na základě pseudonáhodné hodnoty. Pozice generovaného bonusu je nastavována taky na pseudonáhodnou hodnotu, ale s uvažováním objektů, kde se vygenerovat nesmí. Ty jsou předány pomocí metody `SetNotGenerateObjects` a pokud vygenerovaný bonus koliduje s nějakým z těchto objektů, je pozice generována znovu. Generovat se nesmí na místě, kde je planeta a taky na aktuálních pozicích lodí. Kraje obrazovky jsou také omezeny.

Pro detekci sebrání bonusu slouží metoda `PowerupCollide`, která předaný objekt obecného typu `GameObject` otestuje na kolizi se všemi aktuálně vygenerovanými bonusy. Pokud nastala kolize, na konkrétní bonus zavolá metodu `OnPickUp` a vrátí jeho atributy (struktura `Attributes`).

5.3.6 Lodě

Tento celek pokrývá chování prvků ovládaných hráči – vesmírných lodí.

ShipResource představuje jednu vlastnost lodě – energii, anebo štíty. V její aktualizační metodě se provádí regenerace hodnoty – vždy po uplynutí určeného intervalu se hodnota zvýší o zadanou konstantu. Třída obsahuje metody pro snížení hodnoty s detekcí, zda překročila minimální hodnotu – zničení lodě apod.

Práci s oběma vlastnostmi současně zapouzdřuje třída `ShipResources`. Umožňuje reset obou vlastností najednou a „přelití“ energie do štítů a naopak.

Ship představuje loď ovládanou hráčem. Dědí od třídy `MovableSpaceObject`, takže na ni působí gravitace a při dosažení kraje obrazovky se ocitne na kraji opačném. Loď používá a interakuje s mnoha objekty probranými výše. Styl jejího pohybu probraný v kapitole 4.4.4 je implementován v aktualizační metodě `Update`. Obsahuje dva generátory částic – jeden pro efekt výbuchu a druhý pro efekt trysek. Jako své atributy dále obsahuje instanci zbraně (třída `Weapon`) a lodních vlastností (třída `ShipResources`). Pro ovládání poskytuje několik metod:

- `Accelerate` pro akceleraci ve směru natočení
- `Turn` pro změnu natočení (ovládání klávesnicí)
- `RotateAtPoint` pro natočení na zadaný bod (ovl. myší, viz obr. 4.6 na str. 34)
- `FireWeapon` pro střelbu z aktuální zbraně
- `TransferEnergyShield` pro přesun energie do štítů a naopak
- `ApplyPowerup` pro aplikaci bonusu
- `Hit` pro zásah se zadaným poškozením

- `Reset` pro reset lodě na zadanou pozici, nebo náhodnou (`ResetRandom`)

Při zásahu z protivníkovy zbraně (`Hit`) se sníží hodnota štítů v `ShipResources` a pokud klesla pod minimum, je zavolána metoda `Explode`, která vyvolá částicový efekt. Pro detekci zničení lodě slouží metoda `IsDestroyed`.

Při vystřelení ze zbraně `FireWeapon` je volána metoda `Fire` třídy `Weapon`, a pokud vrátí `false`, značí to, že aktuální energie nestačila na výstřel z bonusové zbraně a aktuální zbraň je nastavena zpátky na kanón.

Metoda `ApplyPowerup` aplikuje na loď atributy bonusu předané v parametru – buď nastaví aktuální zbraň pomocí metody `SetType` třídy `Weapon`, anebo upraví hodnotu lodních vlastností.

5.3.7 Hráč a ovládání

Informace o jednom hráči jsou uloženy ve třídě `Player`. Ukládá hodnotu jeho aktuálního skóre a obsahuje instanci jím ovládané vesmírné lodě (třída `Ship`).

Ovládání obou hráčů zapouzdřuje třída `Controls`. Obsahuje jeho aktuální konfiguraci a umožňuje s ní dále pracovat. To znamená editovat ji, načítat implicitní hodnoty či testovat přislušné akce konkrétního hráče. Třída úzce spolupracuje s třídou `Menu` při změnách konfigurace ovládání. Pro ukládání a načítaní nastavení z konfiguračního souboru je používáno třídy `ControlsSettings`.

5.3.8 Grafické uživatelské rozhraní

Objekty v tomto celku obsahují výpis textů na obrazovku, implementaci ukazatelů stavu (*progress bar*) a úpravu konfigurace pomocí herní nabídky (menu).

TextObject je třída pro výpis textů na obrazovku. Pomocí předaného písma (třída `SpriteFont`) umí na zadanou pozici vykreslit zadaný text, který si nese ve svém atributu (nastavení metodou `SetText`). Umožňuje také zjišťovat velikost svého textu v pixelech, včetně uvážení měřítka. Metodami `CenterText` lze text na základě jeho velikosti vycentrovat na zadanou pozici. Díky těmto vlastnostem jde text zarovnávat či centrovat na zadaná místa.

Z této třídy dědí objekt `StatusText`, který umí zadaný text zobrazit s časovaným přechodem do cílové barvy.

ProgressBar implementuje jeden ukazatel stavu. Je vykreslován pomocí dvou obdélníkových tvarů `Shape` – ohraničení (pouze čáry) a výplň. Přepočet hodnoty zobrazované veličiny `m_currentValue` na velikost délky výplně se provádí v metodě `UpdateDstPoint`, a to na základě vztahu 4.24 z kapitoly 4.10.2.

Menu představuje veškerou funkčnost herní nabídky. Při jeho aktivaci metodou `SetActive` je vykresleno, hra se přestane aktualizovat (přestanou se volat metody `Update` herních objektů) a veškeré vstupy z klávesnice či myši jsou předávány pouze jemu (metody `PassKeyDown`, `PassMouseDown` a `PassMouseMove`). Díky tomu jej lze ovládat jak myší, tak klávesnicí, a lze měnit nastavení jednotlivých kláves a tlačítek, protože si uchovává odkaz na objekt spravující nastavení ovládání (třída `Controls`).

Zobrazování jednotlivých podnabídek je řešeno pomocí třídy `Submenu`, která položky jedné podnabídky reprezentuje. Každý objekt této třídy si uchovává vektor

(`std::vector`) položek `TextObject`, které představují jednotlivé popisky nabídky. Tyto položky jsou v metodě `Draw` vykreslovány na zadanou pozici, a to pod sebe, s danými vertikálními rozestupy a horizontálně vycentrované. Objekt si také uchovává informaci o právě vybrané položce a kolem ní pak vykresluje čárový obdélník znázorňující její výběr. Pro posun výběru v nabídce poskytuje metody `Up` a `Down` pro klávesnici a `SelectByMouse` pro výběr na základě pozice myši nad nabídkou. Aktuálně vybranou položku lze pak ze třídy `Menu` zjistit pomocí metody `Selected`, která vrací výčet `ItemID`.

Ve třídě `Menu` pak probíhá práce s těmito podnabídkami. Pro jejich přepínání je využito ukazatele `m_actSubmnu`, který uchovává adresu právě aktuální podnabídky. Tím je přepínání dosaženo velmi jednoduše. Při potvrzení položky vybrané v aktuální podnabídce je volána metoda `OnMenuSelect`, která provede na základě výběru příslušnou akci.

PlayerGUI zapouzdřuje veškeré prvky grafického uživatelského rozhraní patřící jednomu hráči. Obsahuje dva ukazatele stavu (energie a štíty hráčem ovládané lodě), jejich textové popisky, zobrazuje aktuálně přiřazenou bonusovou zbraň, textové hodnoty skóre a textovou identifikaci hráče. Na základě předaných souřadnic pak rozhraní vykreslí buď na levou (hráč 1) nebo pravou (hráč 2) stranu. Zobrazované informace o hráči a stavu jeho vesmírné lodě zjišťuje na základě předané reference na objekt třídy `Player`.

5.3.9 Nastavení

Tato část pokrývá nastavení hry. Jednotlivé třídy dědí z abstraktní třídy `Settings` a představují celkem tři konfigurační soubory. Ukládaná nastavení jsou popsána v kapitole 4.11.

GameSettings pokrývá konfiguraci herních vlastností. Jsou uloženy v souboru `game.cfg`.

ColorSettings pokrývá konfiguraci barev. Používá soubor `colors.cfg`.

ControlsSettings pokrýva konfiguraci vstupních zařízení (klávesnice a myši). Je použit soubor `controls.cfg`.

5.3.10 Spolupráce objektů

Základní interakce mezi herními objekty zajišťuje jedna hlavní třída – `RetroSpaceGame`. Je odvozena od základní bázové třídy frameworku `Game`. Veškerá funkcionalita je v překrytých metodách bázové třídy, které rozšiřuje o novou funkčnost. Jako své atributy obsahuje většinu výše zmíněných objektů, pomocí nichž tvoří logiku hry. Jednotlivá rozšířená funkcionalita překrytých metod je následující:

- V metodě `BeforeCreate` je možné upravit počáteční nastavení frameworku. Zde se pouze nastaví titulek okna.
- Poté následuje metoda `Initialize`, ve které se provádí veškerá inicializace herních objektů. Nejprve se načtou nastavení a poté se načtou písma pro výpisy textů (`SpriteFont`). Následuje nastavení atributů lodí, hráčů, planety, pozadí, systému bonusů, zbraní, ovládání a herní nabídky. Nakonec se zavolá metoda `OnResize`, která je probrána dále.

- Metoda `Resize` je volána při příchodu události o změně velikosti okna. Zavolá pouze metodu `OnResize`, ve které jsou veškeré akce reagující na tuto změnu. Provádí se v nich centrování objektů, aby odpovídali nové velikosti okna. Na začátku metody je upravena projekční matice, aby její pohledový objem odpovídal novým rozměrům okna. Poté se centrují texty, ukazatele, planeta, menu, přegenerovává se pozadí apod.

  Samostatná funkce `OnResize` byla vytvořena proto, aby se tyto veškerá vycentrování mohla provést i hned po startu aplikace, ne jen po příchodu události o změně okna (po startu aplikace se tato událost totiž nevygeneruje).

- V metodě `KeyDown` se provádí reakce na stisk kláves, které se mají provádět jen při stisku, ne držení. Tzn. aktivace menu, střelba a přesun energie. Zjišťování, jaká akce a kterého hráče se má provést probíhá pomocí metody `KeyboardAction` třídy `Controls`. V případě aktivního menu se mu stisknutá klávesa předá metodou `PassKeyDown` a ostatní akce se neprovádí. Akce na základě návratu z menu provádí metoda `HandleMenuResult` – ukončí hru anebo vyvolá novou.

- `MouseDown` provádí to samé co `KeyDown`, jen pro tlačítka myši.

- `MouseMove` provádí akce pří pohybu myši, tzn. otáčení lodě a označení v menu.

- V metodě `Update` se provádí veškerá logika hry. Nejprve se pomocí metody `HandleInputUnbuffered` provedou akce klávesnice a myši, na které má být reagováno během jejich držení – akcelerace a otáčení (opět na základě instance třídy `Controls`). Následuje aktualizace lodí (metody `Update`), systému bonusů a planety. Poté je volána metoda `HandleCollisions`, která provede detekci kolizí mezi objekty a na jejím základě aktualizuje jejich stav.

  V této metodě se se nejprve zkontrolují kolize mezi planetou a projektily (`ProjectileCollide`), aby při nárazu mohly být zničeny. Poté se provede zpracování kolizí mezi loděmi a bonusy, následně planetou a loděmi a nakonec mezi projektily a loděmi. V případě, že je loď zasažena, volá se její metoda `Hit` se zjištěným poškozením.

  Po správě kolizí následuje otestování, jestli nějaká z lodí nebyla zničena (metoda `IsDestroyed`). Pokud ano, aktualizuji se hodnoty skóre, vypíše se text (třída `StatusText`) s informací o tom, kdo skóroval a je vytvořeno nové kolo (metoda `NewGame` – reset vystřelených pojektilů, lodí a systému bonusů). Nakonec se aktualizuje rozhraní hráčů (`PlayerGUI`) a stavový text.

  V případě, že je aktivní menu, se aktualizace objektů neprovádí – hra je zapauzovaná.

- Metoda `Draw` slouží jen pro vykreslení aktuálního stavu hry. Nejprve se vyresetuje vykreslovací část a následně jdou volány metody `Draw` všech objektů, které chceme vykreslit (s parametrem instance třídy `Renderer`, která se z bázové třídy frameworku získá pomocí metody `GetRenderer`). Na konci metody je provedena metoda `SwapBuffers` pro aplikaci double-bufferingu.

- V metodě `Deinitialize`, která je volána při ukončování programu, se provede zrušení načtených písem a uloží se aktuální nastavení do konfiguračních souborů.

Pro spuštění celé hry pak stačí ve vstupní funkci programu `main` (soubor `Main.cpp`) vytvořit instanci této třídy a zavolat na něj metodu `Run`. Návrat z této metody poté značí, že hra skončila a celý program může také skončit.

5.4 Překlad a přenositelnost

Pro úspěšný překlad je nutno překladače jazyka C++ a nainstalovaných knihoven SDL a OpenGL. Aplikace přímo nepoužívá žádné, platformě závislé funkce operačního systému, pouze funkce ze standardních systémových knihoven C++ a knihoven zmíněných výše, které jsou přenositelné mezi platformami. Proto je aplikace také přenositelná, a to na platformy, kde existuje překladač jazyka C++ a implementace těchto knihoven.

Překlad je možný pomocí přiloženého souboru `Makefile` za použití programu GNU Make a překladače G++. Na platformě Windows lze překlad provést také na základě přiloženého projektu do Microsoft Visual Studia 2010.

Kapitola 6

Testování

U každého vyvíjeného počítačového programu hraje důležitou roli testování, protože při vývoji je prakticky pravidlem, že vznikne mnoho chyb. U her toto platí dvojnásob. Nestačí pouze základní testy, jestli program neobsahuje chyby a odpovídá zadání, důležitá je také zpětná vazba od řadových uživatelů, protože především pro ně je hra od počátku vyvíjena. Proto také testování probíhalo ve dvou fázích.

V první fázi testování bylo nejprve nutno odhalit základní chyby v implementaci. Toho se dosáhlo pravidelným hraním a zkoušením různých, neobvyklých situací. Po opravení nalezených chyb se přešlo k druhé fázi.

V druhé fázi byly testovány herní principy se zvolenou konfigurací a celkový grafický dojem. Protože tyto vlastnosti nelze objektivně posoudit jen z pohledu jednoho člověka, navíc ještě vývojáře, bylo pro testování využito hodnocení na základě dojmů většího počtu uživatelů.

6.1 Uživatelské hodnocení

Hra byla poskytnuta pro vyzkoušení celkem čtrnácti uživatelům ve věkovém rozmezí 17-49 let. Ti měli za úkol si hru zahrát a pak do dotazníku shrnout své dojmy. Uživatelé měli k dispozici manuál, který popisoval základní herní principy a možnosti konfigurace.

Dotazník byl cílen na dva hlavní prvky – celkový vizuální dojem (vzhled, efekty apod.) a hratelnost.

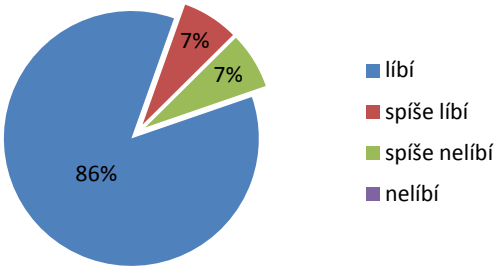
6.1.1 Vizuální dojem

Tato část dotazníku se skládala ze dvou položek – celkového hodnocení a volitelné možnosti napsat svůj konkrétní názor. Výsledek celkového hodnocení je zachycen v grafu na obrázku 6.1.

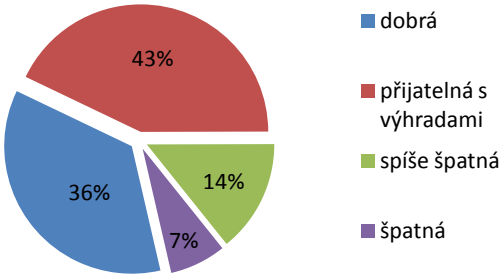
Z grafu lze vidět, že naprosté většině se vizuální dojem ze hry líbil. Konkrétní připomínka byla pouze jedna, a to že by nebylo naškodu přidat další, pokročilejší efekty, za použití shaderů. Toto obohacení se plánuje do dalších verzí (viz kap. 7.1). V této verzi tedy nebylo třeba vzhled na základě hodnocení jakkoli upravovat.

6.1.2 Hratelnost

Hodnocení hratelnosti se skládalo také ze dvou položek – celkového hodnocení a volitelných připomínek. Výsledek celkového hodnocení je zachycen v grafu na obrázku 6.2.



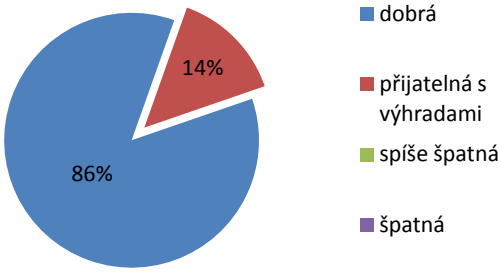
Obrázek 6.1: Hodnocení celkového vizuálního dojmu uživateli.



Obrázek 6.2: Prvotní hodnocení hratelnosti uživateli.

Z obrázku lze vidět, že oproti hodnocení vizuálního dojmu už graf není tak jednoznačný. Konkrétních připomínek bylo více, ale všechny mířily na jedno místo – generování bonusů. Během hry se často stávalo, že oběma lodím došla energie a bylo nutno čekat na vygenerování bonusu, protože měl nastavené moc velké časové rozestupy. Tyto rozestupy jdou sice měnit v rámci konfigurace, ale mít vyladěné prvotní nastavení je velmi důležité, protože mnoho uživatelů to pak může od hry odradit.

Rozestupy byly tedy zmenšeny a hra se dala znovu otestovat. Výsledek tohoto hodnocení lze vidět v grafu na obrázku 6.3.



Obrázek 6.3: Hodnocení hratelnosti uživateli po upravení konfigurace.

Vidíme, že po této úpravě se hratelnost velmi zlepšila a konfigurace byla vyladěna na optimální hodnoty.

U her je tedy důležité vyladit počáteční nastavení, protože většina uživatelů dá hlavně na první dojem.

Kapitola 7

Závěr

Cílem této práce bylo vytvořit počítačovou hru pro dva hráče inspirovanou jednou z prvních počítačových her vůbec – vesmírnou hrou Spacewar!. Motivace spočívala v připomenutí si tohoto zábavného konceptu z počátku herní historie i v dnešní době, a to jak hráčům starším, kteří na podobných hrách vyrůstali, tak i hráčům mladším, kteří s tímto konceptem nepřišli do styku.

Úvodní část práce rozebírá počítačové hry včetně analýzy podobných titulů tohoto žánru a dále hardwarově akcelerovanou počítačovou grafiku, která je při vývoji hry použita.

Při návrhu byl kladen důraz na retrospektivní vzhled v moderním stylu a na jednoduchou, ale zábavnou hratelnost. Těchto prvků bylo dosaženo grafikou složenou většinově pouze z čar, za použití jejich vyhlazení a průhlednosti, a pomocí různých efektů, především částicových. Hratelnost byla obohacena větším počtem zbraní, systémem bonusů, uvažováním energie a štítů u lodí a planetou působící gravitací jak na lodě, tak i na projektily zbraní. Ovládání hry bylo navrženo plně konfigurovatelné, a to buď pomocí klávesnice, anebo myši.

Aplikace byla vyvinuta za použití jazyka C++ a multiplatformních knihoven OpenGL a SDL, takže je zaručena přenositelnost, která byla také úspěšně otestována. Důraz byl kladen na objektový navrh a rozdělení aplikace do několika logických celků, aby byla v budoucnu jednoduchá udržovatelnost a rozšiřitelnost, která je důležitá pro plánovaná rozšíření, zmíněná v části 7.1.

Hra byla úspěšně otestována na operačních systémech Windows (verze XP a 7) a Linux (distribuce Ubuntu 10.04 a CentOS 5), a to na 32 i 64 bitových verzích. Pro vyladění konfigurace hry bylo využito zpětné vazby na základě hodnocení uživatelů. Zadání práce tedy bylo splněno v plném rozsahu.

Ve srovnání se stávajícími tituly podobného žánru (diskutovaných v kapitole 2.4.1) hra nabízí podobný retrospektivní vzhled, ale modernějšího stylu a s využitím částicových efektů. Dále obsahuje jedinečnou kombinaci herních prvků, které tímto stylem ještě nebyly použity. Tím hra přináší novou formu hratelnosti a tak může být atraktivní i pro hráče, kteří se stávajícími tituly již mají zkušenost. Ovládání pomocí myši také není v těchto hrách příliš obvyklé. Pomocí něj lze hru ovládat efektivněji, což pak ještě více zvyšuje její hratelnost.

7.1 Možnosti rozšíření

Jako rozšíření do budoucna se nabízí mnoho možností. V prvé řadě připadá v úvahu vytvoření více zbraní a bonusů. Pro zajímavější hratelnost by se daly zahrnout různé další herní prvky – například volně se pohybující zničitelné asteroidy po hrací ploše či možnost přidání hráče ovládaného počítačem (umělou inteligencí). Velké oživení hratelnosti by určitě přinesla také možnost síťové hry, a to i při více než dvou hráčích.

V rámci zlepšení herního zážitku by bylo velmi efektní přidání pokročilejších grafických efektů, například s využitím shaderů – znázornění deformací, rozmazání obrazu, či částice generované přímo grafickou kartou. Lepšího dojmu by se také mohlo dosáhnout přidáním zvukových efektů.

Velmi zajímavá je i myšlenka převést hru na nějakou mobilní platformu, například *Android*, na které by mohla, s podporou síťové hry, oslovit další okruh hráčů.

Literatura

[1] Baker, S.: Keyboards Are Evil. [online], Last modified on 18 December 2007. [cit. 2011-01-31].
URL http://www.sjbaker.org/wiki/index.php?title=Keyboards_Are_Evil

[2] Blythe, D.: Line and Point Antialiasing. [online], 1999-08-06 [cit. 2011-05-14].
URL http://www.opengl.org/resources/code/samples/sig99/advanced99/notes/node120.html

[3] D'Agata, G.; Molofee, J.: NeHe Lesson 17: 2D Texture Font. [online], [cit. 2011-01-31].
URL http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=17

[4] Doswa: Line Segment to Circle Collision/Intersection Detection. [online], Posted on 2009-07-13 [cit. 2011-04-16].
URL
http://doswa.com/blog/2009/07/13/circle-segment-intersectioncollision/

[5] EDuke32: Duke3D for Windows, Linux and OS X. [online], [cit. 2011-01-31].
URL http://eduke32.com/

[6] Ericson, C.: *Real-time collision detection*. Elsevier, 2005, ISBN 978-1558607323.

[7] GLUT: The OpenGL Utility Toolkit. [online], [cit. 2011-01-30].
URL http://www.opengl.org/resources/libraries/glut/

[8] Hawkins, K.; Astle, D.: *OpenGL game programming*. Game development series, Prima Tech, 2001, ISBN 978-0-7615-3330-6.

[9] Kršek, P.: Základy počítačové grafiky. Studijní opora, FIT VUT, Brno, 2006.

[10] Kršek, P.: Základy počítačové grafiky – Grafická API a OpenGL. Slidy k přednášce, FIT VUT, Brno, 2009.

[11] Minařík, P.: Detekce kolizí v DirectX (collision detection). [online], 23.03.2004 [cit. 2011-04-16].
URL http://programovani.net-mag.cz/?action=art&num=459

[12] MSDN: XNA Developer Center. [online], [cit. 2011-02-23].
URL http://msdn.microsoft.com/cs-cz/xna/

[13] OpenGL Software Development Kit: OpenGL 2.1 Reference Pages. [online], [cit. 2011-02-16].
URL http://www.opengl.org/sdk/docs/man/

[14] Pecinovský, R.: *Návrhové vzory*. Brno: Computer Press, 2007, ISBN 978-80-251-1582-4.

[15] Poláček, P.: World of Warcraft baví 12 milionů hráčů, co bude dál? [online], 7.10.2010 [cit. 2011-01-31].
URL http://games.tiscali.cz/byznys/world-of-warcraft-bavi-12-milionu-hracu-co-bude-dal-53894

[16] Poslušný, R.: Historie počítačových her. [online], [cit. 2011-02-23].
URL http://www.fi.muni.cz/usr/jkucera/pv109/xposlusn.html

[17] Rylich, J.: Dan Vávra: Jak se dělá hra. [online], 10.3.2007 [cit. 2011-01-30].
URL http://pc.hrej.cz/clanky/dan-vavra-jak-se-dela-hra-1078/

[18] Seddon, C.: *OpenGL Game Development*. Wordware Publishing, 2005, ISBN 978-1556229893.

[19] Sláma, D.: Chléb a hry: Historie počítačových her. [online], 6.7.2009 [cit. 2011-01-31].
URL http://www.zive.cz/clanky/chleb-a-hry-historie-pocitacovych-her/sc-3-a-147762/default.aspx

[20] Turek, M.: SDL: Hry nejen pro Linux (1). [online], 22.2.2005 [cit. 2011-01-30].
URL http://www.root.cz/clanky/sdl-hry-nejen-pro-linux-1/

[21] Underdogs: SpaceWar. [online], [cit. 2011-04-20].
URL http://www.homeoftheunderdogs.net/game.php?id=3916

[22] Wikipedia: Computer Space. [online], Last modified on 12 April 2011 [cit. 2011-04-28].
URL http://en.wikipedia.org/wiki/Computer_Space

[23] Wikipedia: Geometry Wars. [online], Last modified on 16 March 2011 [cit. 2011-04-27].
URL http://en.wikipedia.org/wiki/Geometry_Wars

[24] Wikipedia: OpenGL. [online], Last modified on 25 January 2011 [cit. 2011-01-31].
URL http://en.wikipedia.org/wiki/OpenGL

[25] Wikipedia: Spacewar! [online], Last modified on 26 April 2011 [cit. 2011-04-29].
URL http://en.wikipedia.org/wiki/Spacewar!

[26] Wikipedia: DirectX. [online], Last modified on 30 January 2011 [cit. 2011-01-30].
URL http://en.wikipedia.org/wiki/DirectX

[27] Wikipedia: Graphics Processing Unit. [online], Last modified on 30 January 2011 [cit. 2011-01-30].
URL http://en.wikipedia.org/wiki/Gpu

[28] Wolf, M. J. P.: *The video game explosion: a history from PONG to Playstation and beyond*. Greenwood Press, 2008, ISBN 978-0313338687.

Seznam příloh

Příloha A

Popis konfiguračních souborů

Pro konfiguraci aplikace jsou použity celkem čtyři konfigurační soubory:

- `framework.cfg` – nastavení zobrazení (tabulka A.2)
- `controls.cfg` – nastavení ovládání (tabulka A.3)
- `game.cfg` – nastavení hry (tabulka A.4)
- `colors.cfg` – nastavení barev (obsahuje pouze položky typu `Color`)

Hodnoty jsou ukládány na základě klíče ve formátu `klíč=hodnota`. Datové typy a formáty možných hodnot jsou znázorněny v tabulce A.1.

Datový typ	**Popis**	**Formát**	**Příklad**
int	celé č., znaméko		-2
uint	celé č., bez zn.		3
float	desetinné č.		0.5
bool	pravda/nepravda	0(false);1(true)	1
string	řetězec		abc
enum	výčet	uint	5
Vector2i	vektor, celý	int,int	-2,3
Vector2f	vektor, desetinný	float,float	5,0.4
TimeSpan	časový údaj [ms]	uint	200
KeyCode	kód klávesy	enum – číselně	5
BtnCode	kód tlačítka	enum – číselně	10
Color	barva	int,int,int,int	127,0,32,16

Tabulka A.1: Popis použitých datových typů.

Klíč	**Význam**	**Dat. typ**	**Implic.**
WindowManager	správce okna	string	SDL
Renderer	vykreslovací knihovna	string	OpenGL
ScreenSize	velikost okna	Vector2i	960,600
Fullscreen	celá obrazovka	bool	0
VSync	vertikální synchr.	bool	1

Tabulka A.2: Konfigurace zobrazení (`framework.cfg`).

Klíč	**Akce**	**Implicitně**	**Popis**
Player1Controller	hráč 1: ovladač	CTRL_MOUSE	myš
Player1MouseAccel	myš: akcelerace	BC_RIGHT	pravé tlačítko
Player1MouseShoot	myš: střelba	BC_LEFT	levé tlačítko
Player1MouseTransfer	myš: přesun energie	BC_MIDDLE	prostřední tlačítko
Player1KeybAccel	kl.: akcelerace	KC_W	W
Player1KeybShoot	kl.: střelba	KC_LSHIFT	levý shift
Player1KeybTurnLeft	kl.: otáčení doleva	KC_A	A
Player1KeybTurnRight	kl.: otáčení doprava	KC_D	D
Player1KeybTransfer	kl.: přesun energie	KC_Q	Q
Player2Controller	hráč 2: ovladač	CTRL_KEYBOARD	klávesnice
Player2MouseAccel	myš: akcelerace	BC_RIGHT	pravé tlačítko
Player2MouseShoot	myš: střelba	BC_LEFT	levé tlačítko
Player2MouseTransfer	myš: přesun energie	BC_MIDDLE	prostřední tlačítko
Player2KeybAccel	kl.: akcelerace	KC_UP	šipka nahoru
Player2KeybShoot	kl.: střelba	KC_RETURN	enter
Player2KeybTurnLeft	kl.: otáčení doleva	KC_LEFT	šipka doleva
Player2KeybTurnRight	kl.: otáčení doprava	KC_RIGHT	šipka doprava
Player2KeybTransfer	kl.: přesun energie	KC_RSHIFT	pravý shift

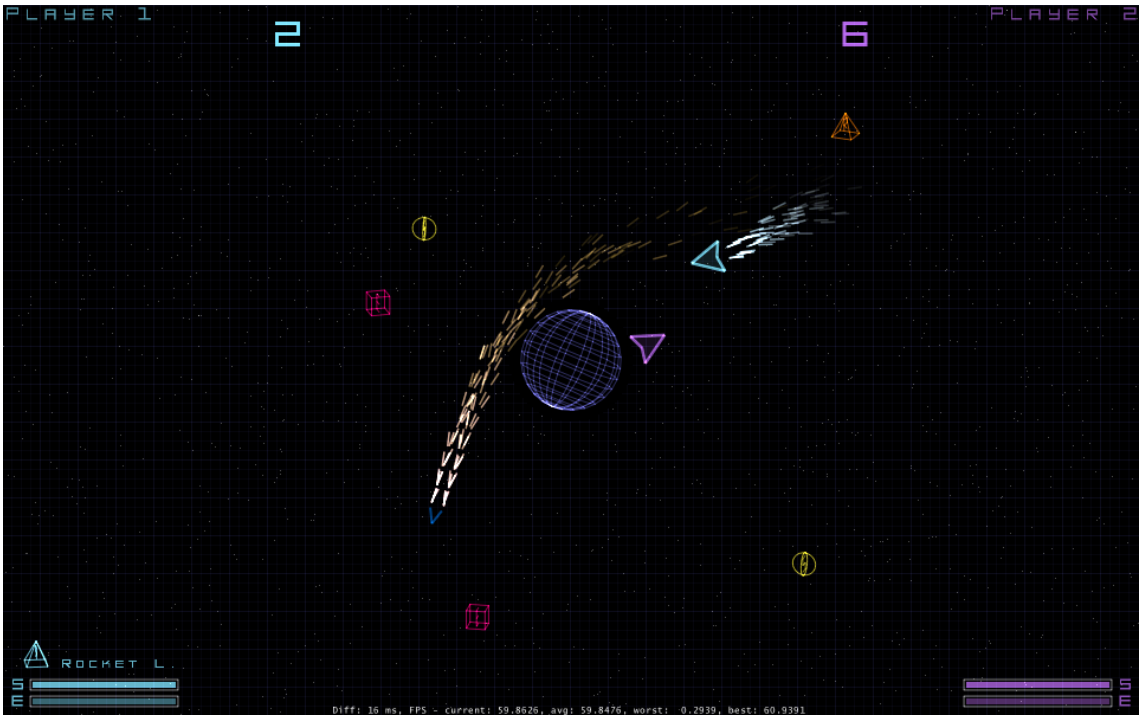
Tabulka A.3: Konfigurace ovládání (`controls.cfg`).

Klíč	**Význam**	**Dat. typ**	**Implic.**
ShipMaxSpeed	max. rychlost lodě	float	0.5
ShipFrictionMult	násobitel zpomalování	float	0.00025
ShipAccelerateSpeed	rychlost akcelerace	float	0.2
ShipTurnSpeed	rychlost otáčení	float	0.3
ShipMaxShield	max. hodnota štítů	float	100
ShipMaxEnergy	max. hodnota energie	float	100
ShipShieldRegenIntervalMs	regen. štítů – interval	TimeSpan	500
ShipShieldRegenValue	regen. štítů – hodnota	float	1
ShipEnergyRegenIntervalMs	regen. energie – interval	TimeSpan	2 000
ShipEnergyRegenValue	regen. energie – hodnota	float	1
PlanetActive	přítomnost planety	bool	1
PlanetGravity	působení gravitace	bool	1
PlanetGravityStrength	síla gravitace	float	1
PowerupMaxSimult	max. poč. bonusů souč.	uint	5
PowerupIntervalMinMs	min. interval gen.	TimeSpan	2 000
PowerupIntervalMaxMs	max. interval gen.	TimeSpan	6 000
PowerupNotGenBorder	vzdál.od kraje kde neg.	Vector2i	25,50
PowerupShieldBonusVal	přidávané štíty	float	35
PowerupEnergyBonusVal	přidávaná energie	float	40
WeaponCannonCooldownMs	kadence kanónu	TimeSpan	100
WeaponCannonEnergyCost	cena energie	float	2.5
WeaponCannonLifetimeMs	doba života	TimeSpan	650
WeaponCannonDamage	velikost poškození	float	5
WeaponCannonVelocity	rychlost	float	0.75
WeaponCannonGravityMultiplier	násobitel gravitace	float	20
WeaponRocketCooldownMs	kadence raket	TimeSpan	800
WeaponRocketEnergyCost	cena energie	float	20
WeaponRocketLifetimeMs	doba života	TimeSpan	1 250
WeaponRocketDamage	velikost poškození	float	30
WeaponRocketVelocity	rychlost	float	0.75
WeaponRocketGravityMultiplier	násobitel gravitace	float	20
WeaponLaserCooldownMs	kadence laseru	TimeSpan	1 000
WeaponLaserEnergyCost	cena energie	float	50
WeaponLaserLifetimeMs	doba života	TimeSpan	50
WeaponLaserDamage	velikost poškození	float	90
WeaponLaserVelocity	rychlost	float	0.001
WeaponLaserGravityMultiplier	násobitel gravitace	float	0
ShowFPS=1	zobrazit FPS	bool	1

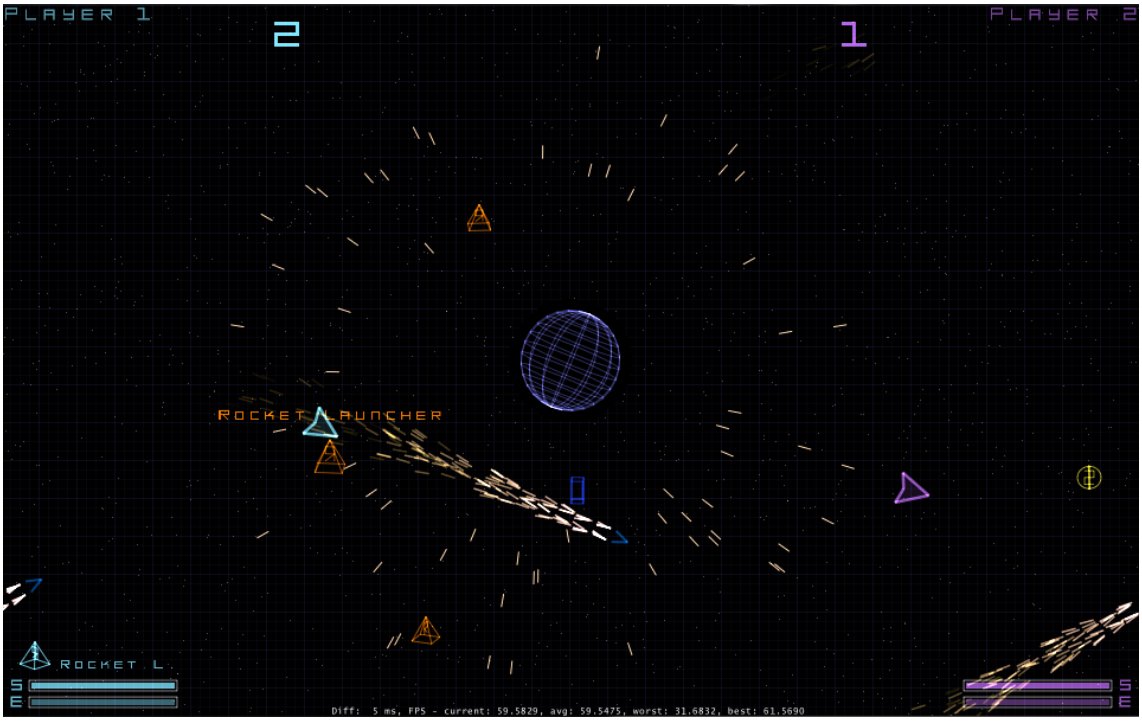
Tabulka A.4: Konfigurace hry (`game.cfg`).

Příloha B

Ukázky aplikace



Obrázek B.1: Probíhající hra se zapnutou gravitací.



Obrázek B.2: Probíhající hra – sebrání bonusu a výbuch rakety.

Obrázek B.3: Herní menu při změně klávesy.